D 2018

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

LEVERAGING METALEARNING FOR BAGGING CLASSIFIERS

FÁBIO HERNÂNI DOS SANTOS COSTA PINTO

TESE DE DOUTORAMENTO APRESENTADA
À FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO EM
ENGENHARIA INFORMÁTICA

Fábio Hernâni dos Santos Costa Pinto

Leveraging Metalearning for Bagging Classifiers

Tese apresentada à Faculdade de Engenharia da Universidade do Porto
para obtenção do grau de Doutor em Engenharia Informática,
realizada sob orientação científica do
Prof. Doutor Carlos Manuel Milheiro de Oliveira Pinto Soares,
Professor Associado da Faculdade de Engenharia da Universidade do Porto,
e co-orientação científica do
Prof. Doutor João Pedro Carvalho Leal Mendes Moreira,
Professor Auxiliar da Faculdade de Engenharia da Universidade do Porto

Departamento de Engenharia Informática
Faculdade de Engenharia da Universidade do Porto

Janeiro de 2018

Aos meus pais,

porque eu escrevi a tese mas eles fizeram tudo o resto.

Abstract

Machine Learning (ML) has been successfully applied to a wide range of domains and applications. One of the techniques behind most of these successful applications is Ensemble Learning (EL), the field of ML that gave birth to supervised learning methods such as bagging, Random Forests or boosting.

The level of expertise required to successfully apply ML techniques to business problems is often very high. This, together with the market scarcity on ML experts, has increased the need for systems that can accelerate and improve the learning process. In this thesis, we focus on how to use metalearning (MtL), the field of ML that studies how learning can be used to solve learning problems, to automate and improve the performance of bagging, one of the most popular EL algorithms.

The scientific contributions of this thesis are split into two parts of this volume: Automated Machine Learning (autoML) in part II and Ensemble Learning in part III. In part II, we extend the state-of-the-art in metalearning and *autoML* with the following contributions: 1) a MtL framework for systematic generation of metafeatures and 2) an *autoML* system that combines MtL with a learning to rank approach to automatically generate a bagging ensemble.

In part III, we make the following contributions: 1) a method that uses MtL to prune bagging ensembles by analysing the data characteristics of the bootstrap samples that are generated and 2) a MtL method to dynamically combine a subset of predictors from an ensemble according to the characteristics of a given test instance.

In both parts, our contributions show that MtL can be an important component of the learning systems of the future. Some of the methods have been published in research papers and therefore, this thesis also serves as a compre-

hensive collection of contributions in a single volume.

Resumo

A área de *Machine Learning* (ML) tem sido aplicada com sucesso numa ampla gama de domínios e aplicações. Um dos conjuntos de técnicas mais bem-sucedidas é *Ensemble Learning* (EL), a área de ML que deu origem a métodos de aprendizagem supervisionada como *bagging*, *Random Forests* ou *boosting*.

O nível de *expertise* necessário para aplicar técnicas de ML a problemas industriais é tipicamente bastante alto. Este facto, juntamente com a escassez no mercado em especialistas em ML, criou a necessidade de sistemas que possam acelerar e melhorar o processo de aprendizagem. Nesta tese, concentramos-nos em como podemos usar *metalearning* (MtL), a área de ML que estuda como a aprendizagem pode ajudar a resolver problemas de aprendizagem, para automatizar ou melhorar o desempenho de *bagging*, um dos algoritmos de EL mais utilizados na indústria.

As contribuições científicas desta tese estão divididas em duas partes deste volume: *Automated Machine Learning* (*autoML*) na parte II e *Ensemble Learning* na parte III. Na parte II, estendemos o estado da arte em *autoML* com as seguintes contribuições: 1) um *framework* de MtL para geração sistemática de *metafeatures* e 2) um sistema *autoML* que combina MtL com uma técnica de *ranking* para afinação automática dos componentes de um modelo *bagging*.

Na parte III, fazemos as seguintes contribuições: 1) um método que usa MtL para fazer *pruning* de modelos *bagging*, através da análise das características da amostras *bootstrap* que são geradas e 2) um método MtL para combinar dinamicamente um subconjunto de modelos de um *ensemble* de acordo para as características de cada instância de teste.

Em ambas as partes, as nossas contribuições mostram que MtL pode ser uma componente importante dos sistemas de aprendizagem do futuro. Alguns dos

métodos foram publicados como artigo científicos e, portanto, esta tese também serve como uma coleção abrangente das contribuições num único volume.

Agradecimentos

Hoje é dia 18 de Janeiro de 2018 e estou sentado no sofá cá de casa a terminar a escrita desta tese que comecei por volta de Outubro de 2013. Foram 4 anos e alguns meses de muito trabalho. Estes agradecimentos vão ser o último texto que acrescento. Enquanto oiço Bowie. A seguir vem The Doors e Nick Drake, segundo a playlist. Não vou começar já a agradecer. Vou começar por expressar o quão farto estou de trabalhar nisto! Tem de ser. Estou tão cansado de corrigir *typos* que vou deixar aqui um de propósitto. Odeio tanto a tese neste momento como adorei fazê-la nestes anos. Yup. Valeu a pena. Aprendi muito mesmo.

Tenho de começar pelo Carlos. Porque de facto começou com ele a convencer-me que conseguia fazer um doutoramento. Eu? Doutoramento? Nunca me tinha passado pela cabeça! No entanto, tive a sorte de ter um orientador de mestrado e doutoramento que é um daqueles raros professores que conseguem transformar um aluno mediano e desmotivado em alguém que acredita que pode chegar ao grupo dos melhores. Talvez pela confiança que passa... sinceramente ainda não percebi muito bem qual é o truque. Lembro-me de pensar "se este gajo acha que posso ser um dos melhores, então é porque posso mesmo!". E isso faz toda a diferença. Mesmo! Ensinas a quem está disposto a aprender e acreditas cegamente no aluno. Não podia pedir mais. Aprendi muito contigo. Não sei se tens noção do enorme impacto que tiveste na minha vida e provavelmente nunca vou conseguir expressar-me de forma a que isso fique claro. Obrigado! Como és também um amigo e um bacano, vamos continuar a ver-nos por aí, nem que seja naqueles concertos manhosos de *indie* do Primavera que só tu gostas. E este pensamento fez-me lembrar que no meio de todas as virtudes tens um enorme defeito: como é possível não gostar de Pink Floyd?!?

Professor João. Outro bacano! E que também está sempre lá, mesmo quando

parece nem ter tempo para respirar. Ajudou-me imenso a pensar e formalizar problemas com rigor científico. Sabe tudo o que há para saber sobre Ensemble Learning, parece que tem uma biblioteca na cabeça. E sempre com uma humildade incrível. Não podia pedir melhor co-orientador também

Há mais duas pessoas que tiveram um impacto direto neste trabalho às quais não posso deixar de agradecer: o Vítor e o Professor Pavel. Ao Bitó por ser aquele gajo tímido que parece que se esforça por passar despercebido mas inevitavelmente vai-se tornar num dos melhores investigadores portugueses em Machine Learning. Para mim foi super importante começar a colaborar contigo nesta fase final da tese, deu-me aquela pica que faltava para conseguir acabar isto bem e como eu queria. Depois do MLj vamos ao JMLR, andamento nisso! Ao Professor Pavel por ser um visionário, mentor e extraordinário investigador. Não me lembro de ter uma conversa consigo em que não tivesse aprendido alguma coisa. Recordo-me agora daquele jantar do ECML de 2016 em que lhe perguntei quando foi a primeira vez que ouviu falar no termo metalearning... "o termo metalearning? Fui eu que inventei! Pois, está claro!" Pfff, estudasses! Claríssimo!

Acho que a tese também se fez pelo contexto em que estava inserido e mais uma vez não podia pedir mais. O LIAAD 1.0 que me recebeu em 2013 há-de ser sempre um dos melhores grupos de investigação que um aluno de doutoramento pode desejar. Do dia para a noite vi-me rodeado de alguns dos melhores jovens investigadores de Machine Learning da minha geração: (liaad = c("Cláudio", "Márcia", "Pedro"); sample(liaad)), Matias, Rafa, Rui, Vânia, Douglas, João, Conceição e Sóninha. Ao LIAAD 2.0/CESE, que fizeram renascer os almoços: Tiago, João, Catarina, Dario, Bruno, Joana, Maria João, Miguel, Kemily. Aos malucos que passaram pelo INESC. À minha equipa na Farfetch, que me ajudou a ser um melhor *cientista dos dados*: João, Lage, Ana, Paula, Carvalheira, Andreia, Antonieta, Otto; à malta que queria ser de recomendações mas não é: Ana, Lia, Ricardo, Baía e Hugo. E ao Carlos e Cristina pela incrível aposta que fizeram em mim!

Aos meus amigos, eles sabem quem são. Aos amigos do rock, por me ajudarem a descarregar energia às quarta-feiras a fazer algo que tanto gosto: José Manuel (!!!), Edjiboy, Srayre e Jonatsu. *Shliffen wacken*!!!

À música. Às minhas guitarras. Aos livros. À Nova Zelândia. À Becky e ao Kiko. Aos que cá não estão mas deviam estar. À piripupi. Aos que irão estar. Aos meus avós. Aos meus pais. À minha família. zzzzzzzzzzzzzz

Acknowledgements

This PhD thesis is partially funded by FCT/MEC through PIDDAC and ERDF/ON2 within project NORTE-07-0124-FEDER-000059, a project financed by the North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF); partially funded by the ECSEL Joint Undertaking, the framework programme for research and innovation horizon 2020 (2014-2020) under grant agreement number 662189-MANTIS-2014-1; and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT), within project UID/EEA/50014/2013.

Contents

List of Figures

List of Tables

Part I

Prologue

Chapter 1

Introduction

The fusion and exponential growth of technologies is giving rise to what some call a Fourth Industrial Revolution [Schwab, 2017]. This process is creating an overlap between the boundaries of the physical and digital world. At the core of this technological turmoil, we find artificial intelligence, machine learning, and more specifically, the ability to learn from data. And today's world is deluged by data.

The dissemination of the Internet around the globe together with the development of ubiquitous information-sensing mobile devices, wireless sensor networks and information store capacity, has enhanced the need to understand and make value of the data that is being generated. Data Science, a recently coined term that brings together statistics, machine learning and computer science, emerges as the field that can assist humans in this task [Miller, 2013]. Research on machine learning plays a central role in the development of this field since it provides the techniques and methods that enable to learn from data.

One of the most common challenges raised by this huge volume of data is focused on supervised learning: the task of generating a function that represents the relationship between a set of variables and a label (classification) or numeric variable (regression). This kind of problem can be seen in multiple applications (finance, retail, banking, industry, to name a few - Han *et al.* [2006]). Several techniques have been proposed for supervised learning. From a predictive performance perspective, ensemble learning (EL) methods are one of the groups

of techniques that present better performance [Zhou, 2012]. EL is a process that uses a set of models (regression or classification), each of them obtained by applying a learning process to a given problem. This set of models is integrated in some way to obtain the final prediction [Mendes-Moreira *et al.*, 2012]. EL has become increasingly popular both for regression and classification tasks. Besides the extensive research that reports great results with ensemble algorithms in a wide variety of problems [Zhou, 2012], data mining competitions with great media coverage (for instance, Netflix prize, Heritage Health prize, among others) proved that ensembles are among the top techniques for predictive modelling.

1.1 Problem Overview

In the past few years, the growth of data volume, velocity and variety has increased dramatically the demand for data scientists, particularly in the industry [Columbus, 2017]. This shortage of data science experts is particularly notorious in specific skills, such as deep learning or EL, since these are the techniques that dominate the state-of-the-art in several domains. Moreover, the lack of data scientists together with the data explosion (which *per se* creates more opportunities for machine learning applications), creates the need for tools that can enhance data scientists' productivity. The resulting research field that aims to answers these needs is automated machine learning (*autoML*), by merging ideas and techniques from several ML and optimization topics, such as Bayesian optimization, metalearning and combinatorial optimization. We will focus particularly on **metalearning**.

Metalearning (MtL), as defined by Brazdil *et al.* [2009], is "the study of principled methods that exploit metaknowledge to obtain efficient models and solutions by adapting machine learning and data mining processes". In this thesis, we study how MtL can be used as a mechanism to automate or improve machine learning processes, specifically, EL methods.

Mendes-Moreira *et al.* [2012] split EL research into three topics: ensemble generation (how to generate the single models that constitute the ensemble), pruning (how to prune an ensemble after its generation to reduce its size and eventually improve performance) and integration (how to combine the single

models that constitute the ensemble to make a final prediction). We adapt the same structure of the literature and we focus on specific problems in each topic:

Ensemble generation. When combining several models for a predictive task, it is difficult to understand the behaviour of the resulting ensemble and why it performs well or not. The models of an ensemble need to be complementary in different regions of the input space but measuring diversity is a very difficult task [Kuncheva and Whitaker, 2003]. Therefore, this has led diversity to become a topic of paramount importance for the EL field, both from a theoretical and a practical perspective [Brown *et al.*, 2005].

Ensemble pruning. Research on EL has shown that it is possible to reduce the size of an ensemble without loss of performance. In some cases, researchers even reported predictive gains [Zhou *et al.*, 2002]. These findings led to the development of several methods for ensemble pruning [Martinez-Muñoz *et al.*, 2009], particularly useful for the scenarios in which the computational resources are scarce [Qian *et al.*, 2015].

Ensemble integration. Using an ensemble to obtain predictions is usually a straightforward process. If we generate an ensemble that our evaluation methodology estimates to be accurate, that model is going to be used to score any instance that we want to predict, regardless of its characteristics. An alternative to this is a dynamic approach: automatically select one or more predictive models from an ensemble according to the characteristics of a given test instance.

From a practical point-of-view, designing an ensemble that takes into account all this stages can be very complex, particularly for a data scientist that is a non-expert in EL. Furthermore, we believe that each one of this stages of EL can be improved by learning from past experience, which often is discarded by EL algorithms.

1.2 Research Question and Contributions

One of the earliest and most influential EL algorithms is bagging [Breiman, 1996a] (pseudo code is provided in section 2.2). Since its also one of the most widely used EL algorithms both in the academic world as in the industry (and

one of the building foundations of Random Forests, which is one of the most popular algorithms these days), we selected it as our object of study. Therefore, in this PhD thesis, we want to test the validity of the following hypothesis:

Is it possible to automate and improve the performance of bagging by using metalearning?

Naturally, the scope of this hypothesis is wide enough to associate it with a possibly infinite number of sub-problems. We do not aim to solve them all. Our main focus is to understand the state-of-the-art in bagging and bagging-related methods focusing on ensemble generation, pruning and integration; identify gaps that can be fulfilled by MtL systems; and finally design, evaluate and propose those systems.

1.2.1 Contributions

The contributions of this thesis are briefly summarised as follows:

- A MtL framework for systematic generation of metafeatures.
- New metafeatures that we show to be informative for characterizing datasets, particularly for the algorithm selection problem.
- An *autoML* system that combines MtL with a learning to rank approach to automatically optimize a bagging ensemble.
- An empirical methodology to study the behaviour of bagging and give insights about the desired properties of a bagging ensemble.
- A method that uses MtL to prune bagging ensembles by analysing the data characteristics of the bootstrap samples that are generated.
- A MtL method to dynamically combine a subset of predictors from an ensemble according to the characteristics of a given test instance.

1.3 Structure of this thesis

The remainder of this thesis is structured as follows:

- **Part I: Prologue.** This part combines chapters 1 and 2 of this thesis, serving as introduction and overview of the topics explored, respectively. Chapter 2 provides a brief overview of the state-of-the-art both for EL and MtL, focusing particularly on the intersection between the two. However, it is important to notice that, for each chapter of this thesis, we provide a specific section in which we detail the related work of each contribution in the chapter.

- **Part II: Automated Ensemble Learning.** This part includes chapters 3 and 4. In the former, we propose a *MtL framework that enables the generation of metafeatures in a systematic fashion.* Although we present and evaluate the framework for algorithm selection in classification problems, the framework can be generalized for other ML problems. In fact, this is accomplished in chapter 4, in which we use the framework together with a learning to rank approach to design an *autoML system that is able to automatically rank bagging workflows.*

- **Part III: Metalearning for Ensemble Learning.** In this part we comprise the chapters that provide contributions for the three sub-fields of EL: generation, pruning and integration. In chapter 5, we propose an empirical methodology for the analysis of the behaviour of the bagging algorithm using MtL. In the same chapter, we use the metadata collected from applying the methodology to introduce an *ensemble pruning technique for bagging ensembles.* Finally, in chapter 6, we design and propose *a MtL system for dynamic integration of ensembles* for classification problems.

- **Part IV: Epilogue.** The final part of the thesis is composed by chapter 7, that concludes the volume summarizing the work done and addressing the research question defined initially. Finally, ideas for future research are suggested.

Chapter 2

Overview

In this section, we describe an overview of the literature for the two main sub-fields of ML that we address in this thesis: EL and MtL. We focus particularly on papers that somehow relate these two sub-fields.

2.1 Error, Accuracy and Diversity in Ensembles

Historically, the two pioneering papers that laid the roots for EL research presented different perspectives on the same subject: Hansen and Salamon [1990] published an empirical work in which was found that predictions made by an ensemble of classifiers (neural networks) are often more accurate than the best single classifier. On the other hand, Schapire [1990] showed theoretically that weak learners can be combined to form a strong learner which settled the basic concept behind Boosting algorithms.

EL literature is very clear about what characteristics a good ensemble must present: the predictors must be *accurate* and *diverse* in order to complement themselves. The concept of complementarity is very important. Combining complementary classifiers can improve the accuracy over individual models. One can say that two classifiers are complementary if they make errors in different regions of the input space [Brown *et al.*, 2005].

The generalization error decomposition for regression ensembles was a very important step in understanding the behaviour of such systems, and its contributions helped to guide the research in ensemble generation [Krogh and Vedelsby,

1995; Ueda and Nakano, 1996]. For classification, there is no such unifying theory. It is well accepted in the Machine Learning community that generating diverse individual classifiers is a good practice to achieve accurate ensembles. Although there are proven connections between diversity measures and accuracy, there is also evidence that raises doubts about the usefulness of such metrics in building ensembles [Kuncheva and Whitaker, 2003]. A complete grounded framework is still missing in this research field.

One can find work in progress trying to adapt the concepts present in regression for classification problems by choosing to approximate the class posterior probabilities [Tumer and Ghosh, 1996; Fumera and Roli, 2005]. However, for some learning algorithms, it is not possible to extract those probabilities: the outputs have no intrinsic order. Literature in this topic is divided into two directions: ordinal outputs and non-ordinal outputs (in which the outputs of the classifiers are taken as probabilities, as mentioned before). We follow Brown *et al.* [2005] very closely.

For *ordinal outputs*, a theoretical framework for analysing a classifier error when its predictions are posterior probabilities was proposed by Tumer and Ghosh [1996]. Some of the assumptions of this work were later lifted by Roli and Fumera [2002] and Fumera and Roli [2003]. However, besides the limitations imposed by the ability of the learning algorithms to output posterior probabilities, some assumptions made by the framework are possibly too strong to hold in practice (such as the identical variance in the error of the classifiers).

For *non-ordinal outputs*, the state of the art still does not provide a satisfying theory. Ideally, one would have an expression that, similarly to the error decomposition in regression, decomposes the classification error rate into the error rates of the individual learners and a term that quantifies their *diversity*.

The lack of an error decomposition for classification in a context of non-ordinal outputs has led to several diversity measures being proposed in the literature: *Disagreement*, *Q-statistic*, *Kappa-statistic*, *Kohavi-Wolpert variance*, to name a few.[1] However, their usefulness has been highly questioned. Kuncheva and Whitaker [2003] showed through a broad range of experiments that the existing diversity measures do not provide a clear relationship between those and

[1] We refer the reader to Zhou [2012] for further details.

the ensemble accuracy. Tang *et al.* [2006] presented evidence that, compared to algorithms that seek diversity implicitly, exploiting diversity measures explicitly is ineffective while constructing strong ensembles. They also showed that diversity measures do not provide reliable information if the ensembles achieve good generalization performance but, at the same time, are highly correlated to average individual accuracies, which is not desirable.

More recently, two new research directions emerged for understanding ensemble diversity in a classification context: "good" and "bad" diversity [Brown and Kuncheva, 2010] and information theoretic diversity [Brown, 2009].

Brown and Kuncheva [2010] adopt the perspective that a diversity measure should be naturally defined as a consequence of two decisions in the design of the ensemble learning problem: the choice of error function and the choice of integration function. More particularly, with a zero-one loss function and majority voting integration scheme. The authors derive a decomposition of the majority vote error into three terms: average individual accuracy, "good" diversity and "bad diversity". The "good" diversity measures the disagreement on instances when the ensemble is correct. The "bad diversity" measures the disagreement on instances when the ensemble is incorrect.

Based on interaction information (a multivariate generalization of mutual information - see Zhou [2012] for further details), Brown [2009] presented a decomposition of the conditional interaction information between a set of predictors and a target variable. His mathematical formulation proposes to decompose classifier ensembles diversity into three components: *relevancy* (the sum of mutual information between each classifier and the target), *redundancy* (measures the dependency, independent to the target variable, among all possible subsets of classifiers) and *conditional redundancy* (measures the dependency among the classifiers given the class label). The main problem of this decomposition is that there is no effective process for estimating the diversity terms. Zhou and Li [2010] provided a mathematical simplification of Brown [2009] contribution and an estimation method for the diversity terms. However, this framework has the disadvantage that assumes linear classifiers as predictors.

2.2 Bagging, Boosting and other EL algorithms

Research in EL has produced some algorithms that due to their effectiveness have been widely adopted by the ML community and even in the industry. This section gives a brief overview of the most popular ones.

Bagging stands for bootstrap aggregating and is due to Breiman [1996a]. This technique plays a central role in Random Forests [Breiman, 2001a], one of the most popular ensemble learning algorithms.

Algorithm 1 shows the pseudo code for the bagging algorithm. Generically, given a data set containing n number of training instances, a sample with replacement b (a bootstrap) of n training instances will be generated. The process is repeated K times and K samples of n training instances are obtained. Then, from each sample, a model $\hat{h}$ is generated by applying a base learning algorithm H. In terms of aggregating the outputs of the base learners and building the ensemble EH, bagging adopts two of the most common ones: voting for classification (the most voted label is the final prediction) and averaging (the predictions of all the base learners are averaged to form the ensemble prediction) for regression [Zhou, 2012]. An interesting feature of bagging is that allows to estimate the precision of the base learners using the out-of-bag examples (the ones that were not selected for training) in each iteration, allowing to compute the error of the bagged ensemble.

input : Data set D = (x_1, y_1), (x_2, y_2), ..., (x_n, y_n)
Base learning algorithm H
Number of predictors K

for $k \leftarrow 1$ **to** K **do**
  $\hat{h}_k$ = H(b_k) %Train learner on bootstrap b

end

output: EH($\hat{h}_1$, ..., $\hat{h}_K$)

Algorithm 1: Bagging pseudo code. Source: Zhou [2012].

Schapire [1990] published a seminal paper in which he theoretically proved that any weak learner is potentially able to be boosted to a strong learner. This concept originated the family of **boosting** algorithms. Shortly, a boosting algorithm works by sequentially train learners and combine their outputs for a final

prediction. However, each learner is forced to focus more on instances poorly predicted by the previous generated learners (if any) by assigning a weight to each instance based in the evaluation error at each iteration. This weight influences then the instances selected for the next iteration. There are several boosting algorithms proposed in the literature [Zhou, 2012], being *AdaBoost* the most influential one [Freund and Schapire, 1997].

Bagging and boosting exploit variation in data in order to achieve greater diversity (and accuracy) in the final predictions. On the other hand, some ensemble methods exploit difference among learners, such as **stacked generalization**, due to Wolpert [1992]. Stacking initializes by generating a set of models from a set of learning algorithms and a dataset. Then, a meta-dataset is generated by replacing each base-level instance by the predictions of the models.[2] This new dataset is then presented to a learning algorithm that relates the predictions of the base-level models with the target output. A prediction from a stacking model is extracted by making the base-level models predict an output, build a meta-instance and feed it to the meta-learner that provides the final prediction. The stacking framework was improved later with important contributions at the level of meta-features extraction and the selection of the meta-level algorithm by Džeroski and Ženko [2004].

Cascade generalization originated from the work by Gama and Brazdil [2000]. Here, the models are used in sequence rather than in parallel as in stacking: the output of the first generated model feeds the second model; the outputs of the first and second model feed the third model, and so on.

Other ensemble methods present in the literature, although with less impact in the research community, are cascading [Alpaydin and Kaynak, 1998], delegating [Ferri *et al.*, 2004] and arbitrating [Ortega *et al.*, 2001]. For further details, see [Zhou, 2012].

[2]This can lead to overfitting. To avoid this problem, is often recommended to exclude the base-level instances from the meta-dataset and train the stack model in new data [Zhou, 2012].

2.3 Ensemble Generation

The first phase of developing an ensemble is model generation. If the models are generated using the same induction algorithm, the ensemble is called homogeneous, otherwise, in case the models are generated using different induction algorithms, the ensemble is heterogeneous.

Higher diversity is expected when developing heterogeneous ensembles, thus, assuring, eventually, a more accurate ensemble [Gashler *et al.*, 2008]. However, obtaining that diversity with different induction algorithms can be more difficult than with just one. Diversity can be achieved either by *manipulating data* or by the *model generation process*. The following subsection discusses methods for each category.

2.3.1 Data manipulation

Data manipulation for ensemble generation can be split into three different subgroups: sub-sampling from the training set, input features manipulation and output targets manipulation. The first consists in using different sub-samples from the training set to generate different models. This method takes advantage of the instability of some learning algorithms [Breiman, 1996b]. Given some randomness of the inductive process of a learning algorithm and its sensitivity to changes in the training set, one can manipulate the generation of models to obtain a diverse ensemble. Two well known ensemble learning techniques that use this method are the already mentioned boosting and bagging.

Several methods were developed for *input features manipulation*, being the most simple one the random feature selection. More complex techniques are noise injection [Matsuoka, 1992], that consists in adding Gaussian noise to the inputs; iterative search methods for feature selection [Wang *et al.*, 2007] and rotation forests [Rodriguez *et al.*, 2006], a method that combines selection and transformation of features using Principal Component Analysis (PCA).

Output target manipulation is a far more uncommon technique. In a regression scenario, Breiman [2000] signs the most important contributions: output noise injection, that essentially consists in adding Gaussian noise to the output variable of the training set; and iterated bagging [Breiman, 2001b]. The

latter consists of initially generate a model and compute its residuals; a second model is generated with the output target being the residuals of the first model; this iterative process is repeated several times to develop the ensemble. Concerning classification problems, the representation of the class labels is manipulated. Examples of these techniques are Error-Correcting Output Code (ECOC - a method that combines several binary classifiers in order to solve a multi-class problem - Dietterich and Bakiri [1995]), Flipping Output (random changes in the labels of some training instances) and Output Smearing (conversion of multi-class outputs to multivariate regression outputs to construct individual learners), both introduced by Breiman [2000].

2.3.2 Model generation

Achieving diversity by model generation manipulation can be done through three techniques: different parameter sets; manipulation of the induction algorithm or final model manipulation.

The vast majority of learning algorithms is sensitive to *parameter changes*. The number of parameters is highly dependent to the selected algorithm. In order to achieve a diverse set of models, one must focus on the most sensitive parameters of the algorithm. Papers on neural networks [Pollack, 1990] and k-nearest neighbours [Yankov *et al.*, 2006] ensemble generation show the effectiveness of this technique.

Approaches for ensemble generation by *manipulation of the induction algorithm* have two main categories: sequential and parallel. In sequential approaches [Rosen, 1996; Islam *et al.*, 2003], the generated models are only influenced by previous ones. The main feature of these techniques is the use of a decorrelation penalty term in the error function of the ensemble to increase diversity. Making use of the decomposition of the generalization error of an ensemble, the training of each network tries to minimize a function that has a covariance component, thus decreasing the generalization error. In parallel approaches, the generation of the models includes an exchange of information and usually is guided by an evolutionary framework [Liu *et al.*, 2000]. Two distinct parallel techniques are the infinite ensemble of Support Vector Machines models (the core idea is to create a kernel that gathers all the possible models in the

hypothesis space - Lin and Li [2005]) and Random Forests, that combines the bagging method with random feature selection on the generated trees.

Model manipulation is a less studied topic. This group of techniques focus on modify a model in some way so that its performance is boosted (for instance, given a set of rules, produced by one single learning process, one can repeatedly sample the set of rules and build n models [Jorge and Azevedo, 2005]).

2.4 Ensemble Pruning

Ensemble pruning consists of eliminating models from the ensemble, with the aim of improving its predictive ability or reducing computational costs. Research on ensemble pruning is divided into two main categories: partitioning-based and search-based approaches. In the latter, the characterization can be more specific according to the nature of the search algorithm that is used: exponential, randomized and sequential. We follow Mendes-Moreira *et al.* [2012] very closely.

2.4.1 Partitioning-based

Partitioning-based methods divide the pool of models into groups and then the one (or more) most representative of that group is selected to be included in the final ensemble. Typically, the partitioning is done using clustering algorithms such as hierarchical agglomerative clustering [Giacinto *et al.*, 2000] or k-means [Lazarevic and Obradovic, 2001].

2.4.2 Search-based

Concerning *search-based* approaches, exponential search pruning refers to the group of algorithms that tries to find the optimal set of k models from a pool of K models to integrate an ensemble. The searching space of this problem is very large and is a NP-complete problem. For small values of k, this can be a good approach [Martínez-Muñoz and Suárez, 2006]. However, in most of the cases, using a very small k gives poor results. Therefore, besides its high computational cost, this approach gives poor results in comparison with other pruning algorithms used in ensembles with larger values of k.

Randomized search pruning algorithms integrate an evolutionary framework in their process to search for a solution that is better than a random one. The GASEN (Genetic Algorithm based Selective ENsemble - Zhou *et al.* [2002]) algorithm presented very promising results in classification problems. Another relevant method in this category is Pareto Ensemble Pruning [Qian *et al.*, 2015]. This method not only shows superior performance than other state-of-the-art approaches but also provides strong theoretical support for its results.

Regarding sequential methods, these can be categorized into three different clusters: forward (if the search begins with an empty ensemble and adds models to the ensemble in each iteration), backward (if the search begins with all the models in the ensemble and eliminates models from the ensemble in each iteration) or combined (if the selection can have both forward and backward steps).

A forward search method that presented good results was Margin Distance Minimization (MDSQ - Martínez-Muñoz and Suárez [2006]). This method belongs to a group of methods based on ordered aggregation [Martinez-Muñoz *et al.*, 2009]. These methods have the ability to order the predictors according to accuracy/diversity that they add to the ensemble.

Backwards and combined methods are more rare. Coelho and Von Zuben [2006] presented methods that used a backwards search algorithm. Mendes-Moreira *et al.* [2006] presented a method that combined both forward and backward steps.

A more detailed description of the state-of-the-art on this topic can be found in chapter 5.

2.5 Ensemble Integration

Ensemble integration focus on how to combine the output of models previously generated for an ensemble in order to obtain one final prediction. We present the literature on this topic by dividing the methods into two groups: static and dynamic. In the former, the weights assigned to each model in the ensemble are a constant value; in the later, the weights vary according to the instance to be predicted. In the dynamic group, we distinguish between methods for selection

or combination of models. In the former, the method is only able to select one predictor from the ensemble. In the latter, the method selects a set of predictors from the ensemble and combines the outputs.

2.5.1 Static

In the case of classification, the most frequent techniques are majority voting (for binary problems, the final prediction is the label that received more than half of the votes; otherwise, the output will be the rejection option, usually the most frequent class), plurality voting (the final prediction is the label with largest number of votes), weighted voting (a weight is assigned to each learner according to its past performance) and soft voting (here, the output of the classifiers is a probability instead of a label).

The most frequent techniques for regression are averaging (given a set of base learners, the final prediction is the average of the predictions made by the learners) and weighted averaging (given a set of base learners, the final prediction is obtained by averaging the outputs of different learners with different weights implying different importance). Usually the weights are estimated given the past performance of the base learners in some validation data.

One of the most well know methods for ensemble integration is stacking [Wolpert, 1992]. This method consists in training a learning algorithm to combine the predictions of the base-level learners. It can be done on the training data (more prone to overfitting) or on validation data. Džeroski and Ženko [2004] proposed a stacking framework for classification. Their results show that their framework is better than selecting the best classifier by cross validation.

Breiman [1996c] presented a regression version of the original stacking framework. To avoid the multicollinearity problem he used ridge regression as the stack model under the constraint that the coefficients of the regression (in other words, the weights for each model in the ensemble) need to be non-negative. Although the results were not great, an important contribution made by Breiman [1996c] is the empirical observation that most of the weights are equal to zero, which reinforces the need for ensemble pruning.

Kuncheva [2002] proposed an hybrid approach that combines selection with a combination approach. The final prediction can be made by only one predictor

if that predictor passes a statistical test to check if it is significantly better than the others. If the predictor does not pass the test, a combination approach is used.

2.5.2 Dynamic selection

The dynamic approach has been receiving an increasing amount of attention in the research community [Mendes-Moreira *et al.*, 2012; Cruz *et al.*, 2018]. The motivation for this technique is that different models in the ensemble may have different performances on different regions of the input space. The technique suggests that given an input X, similar data is selected from a validation set. This process is usually guided by some distance metric, like the Euclidean distance with the k-nearest neighbours algorithm. Then, one (*selection*) or more models (*combination*) are selected from the ensemble given their past performance on the similar data. If several models are selected, the predictions need to be combined in some way to make the final prediction.

The first paper concerning dynamic selection of classifiers is due to Ho *et al.* [1994]. In this work, the authors proposed a selection based on a partition of training examples. The individual classifiers are evaluated on each partition to find the best one for each. Then, the test instance to be predicted is categorized into a partition and classified by the corresponding best classifier.

Woods *et al.* [1997] proposed two methods that are often used as benchmark in comparative studies [Britto *et al.*, 2014], the DS-LA LCA-based method and the DS-LA OLA-based method. For abbreviation purposes we will refer to these as OLA and LCA, respectively. Both methods calculate an estimation of accuracy of the base classifiers in the local region of the feature space close to the test instance in the training dataset. In OLA, it is computed the percentage of the correct recognition of the samples in the local region; in LCA, it is computed the percentage of correct classifications within the local region, but considering only those examples where the classifier has given the same class as the one it gives for the test instance. In both methods, only one classifier is selected for the final prediction.

Yankov *et al.* [2006] proposed a method to select from an ensemble of two k-NN models the one most suited for a given instance. The selection is done by

Support Vector Machine model using metafeatures extracted from the instances.

Todorovski and Džeroski [2003] proposed the meta decision trees, a MtL method to select the best predictor of an ensemble of decision trees for a given test instance. A more detailed description of this method is given in section 2.6.2.

2.5.3 Dynamic combination

The dynamic combination approach was introduced by Merz [1996]. Results showed that a simple majority combination was superior to their dynamic approach. Woods *et al.* [1997] used a very similar approach but the results (with 4 different datasets) were slightly better.

Kuncheva *et al.* [2007] proposed a method based on the *oracle* concept. Essentially, each classifier of the ensemble consists in two sub-classifiers and a oracle that decides which of the two sub-classifiers is going to be used to predict the test instance. In their work, the oracle is a random linear function. Kuncheva *et al.* [2007] claim that the random oracle idea works because it adds diversity to the ensemble. Ko *et al.* [2008] developed this idea by adding a k-nearest neighbours approach and proposed the KNORA-E and KNORA-U methods. In the former, only the classifiers that correctly classify all the k-nearest patterns are used; in the latter, the classifiers that correctly classify any of the k-nearest neighbours are used - a single classifier can be selected more than once.

Tsymbal [2000] and Tsymbal and Puuronen [2000] combined dynamic integration with classifier ensembles using bagging and boosting algorithms. Results suggest that dynamic integration improves significantly the performance of the ensembles instead of the more typical majority voting integration. Tsymbal *et al.* [2006] also presented experiments in which a dynamic integration approach instead of the simple majority combination in Random Forests was better on some datasets.

Santana *et al.* [2006] proposed a method that explicitly used accuracy and diversity to select a subset of classifiers. The method sorts the classifiers in decreasing order of accuracy and in increasing order of diversity. They presented two versions: DS-KNN, very similar LCA and OLA but it takes into account

diversity; and DS-Cluster, that uses a clustering process to divide the validation set into clusters where the most promising classifiers will be associated.

Liyanage *et al.* [2013] proposed a dynamically weighted ensemble classification (DWEC) framework whereby an ensemble of multiple classifiers are trained on clustered features. The decisions from these multiple classifiers are dynamically combined based on the distances of the cluster centres to each test data sample being classified. Results showed that their method is significantly better than a Support Vector Machine baseline classifier.

Ko *et al.* [2008] and Mendes-Moreira *et al.* [2009] presented studies in which several variants of dynamic selection and combination are experimented. The former, showed comparisons of dynamic ensemble selection and dynamic ensemble combination; results (no statistical verification was carried) suggested that using weak classifiers, the dynamic ensemble combination can marginally improve the accuracy, but not always performs better than dynamic classifier selection. The later, in a regression task, also found evidence that selecting dynamically several models for the prediction task increases prediction accuracy comparing to the selection of just one model. They also claim that using similarity measures according to the target values improves results.

Rooney *et al.* [2004a] extended the dynamic integration for regression problems. They claim that dynamic integration techniques are as effective for regression as stacked regression when the base models are simple. In another paper from the same authors Rooney *et al.* [2004b], they combined the random subspace method (training data is transformed to contain different random subsets of the variables) with stacked regression and dynamic integration. Again, for simple models like linear regression and k-nearest neighbours, these techniques are more effective than bagging and boosting. Later, Rooney and Patterson [2007] proposed a combination of stacking and dynamic integration for regression problems named wMetaComb.

Cruz *et al.* [2015] proposed a method that uses MtL for dynamic combination of classifiers. We provide more details about this method in section 2.6.2.

The nature of the dynamic approach suggests that it is well suited for data streams environment, especially in the presence of concept drifts [Gama *et al.*, 2014]. Some well known data streams techniques use a dynamic approach, such

as the Dynamic Weighted Majority [Kolter and Maloof, 2007] or DDD [Minku and Yao, 2012]. However, for batch problems, comparative studies show that none of the dynamic approaches proposed in the literature dominates the other approaches. Moreover, a recent survey on the topic brought attention to the importance of a method to infer if a given dataset/ensemble benefits from a dynamic method [Britto *et al.*, 2014]. On the same paper, it was found evidence that simpler selection schemes (such as KNORA and LCA) may provide similar, or sometimes even better, classification performances than the sophisticated ones.

2.6 Metalearning

MtL researchers have been mostly concerned with the algorithm recommendation problem, originally formulated by Rice [1976]. Rendell *et al.* [1987] and Rendell and Cho [1990] published the first papers in which the expression "metalearning" is used in Machine Learning. In following years, research in MtL was boosted by two European projects: StatLog and METAL. The former provided an assessment of the strengths and weaknesses of several classification techniques, while the later focused on the development of a MtL assistant for providing user support in machine learning and data mining, both for classification and regression problems [Smith-Miles, 2008].

Early on, researchers identified that the key issue in MtL is metaknowledge: the experience or knowledge gained from one or more data mining tasks. Typically, this knowledge is not generally available to improve the task or to assist in the following data mining tasks. Therefore, MtL focus on the effective application of knowledge about learning systems to understand and improve their performance.

Figure 2.1 illustrates a step by step MtL framework for algorithm selection. The process starts with a collection of datasets and learning algorithms. For each of those datasets, we extract metafeatures that describe their characteristics (A). Then, each algorithm is tested on each dataset and its performance is estimated (B). The metafeatures and the estimates of performance are stored as metadata. The process continues by applying a learning algorithm (a meta-

learner) that induces a meta-model that relates the values of the metafeatures with the best algorithm for each dataset (C). Given the metafeatures of a new dataset (D), this meta-model is used to recommend one or more algorithm for that dataset (E).

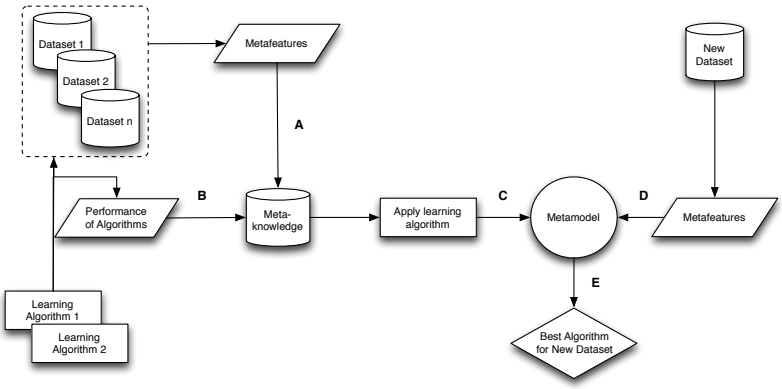

Figure 2.1: Metalearning framework for algorithm recommendation.

2.6.1 Metadata

Generating the metadata is the most important step in a MtL process. Besides choosing the appropriate metatarget for the task, it is crucial to select meaningful *metafeatures* that contain information to successfully achieve the main goal. We provide a more in-depth discussion on metafeatures in section 3.2 of this thesis.

Metafeatures can be divided into three types (Figure 2.2):

- *Simple, statistical and information-theoretic.* These are the most common type of metafeatures extracted using descriptive statistics and information-theoretic measures. Some examples: number of features/examples, number of instances with missing values (simple); mean skewness of numeric features, mean value of correlation (statistical); class entropy, mutual information for symbolic features (information-theoretic). We refer the reader to the following papers for more examples: [Köpf *et al.*, 2000; Gama and Brazdil, 1995].

- *Model-based.* This type of metafeatures are extracted based on properties

of the induced model. For example, the number of leaf nodes in a decision tree [Peng *et al.*, 2002a] or mean of the off-diagonal values of a kernel matrix in a Support Vector Machines model [Soares and Brazdil, 2006].

- *Landmarkers.* This type of metafeatures are quick estimates of an algorithm's performance. They can be obtained in three different ways: through the run of simplified versions of an algorithm (for instance, a decision stump [Bensusan and Giraud-Carrier, 2000; Pfahringer *et al.*, 2000]); quick performance estimates on a sample of the data, also called sub-sampling landmarkers [Fürnkranz and Petrak, 2001]; and finally, through an ordered sequence of sub-sampling landmarkers for a single algorithm, which allows to form the so called learning curve of an algorithm. In this case, not only the estimates can be used as metafeatures but also the shape of the curve [Leite and Brazdil, 2005].

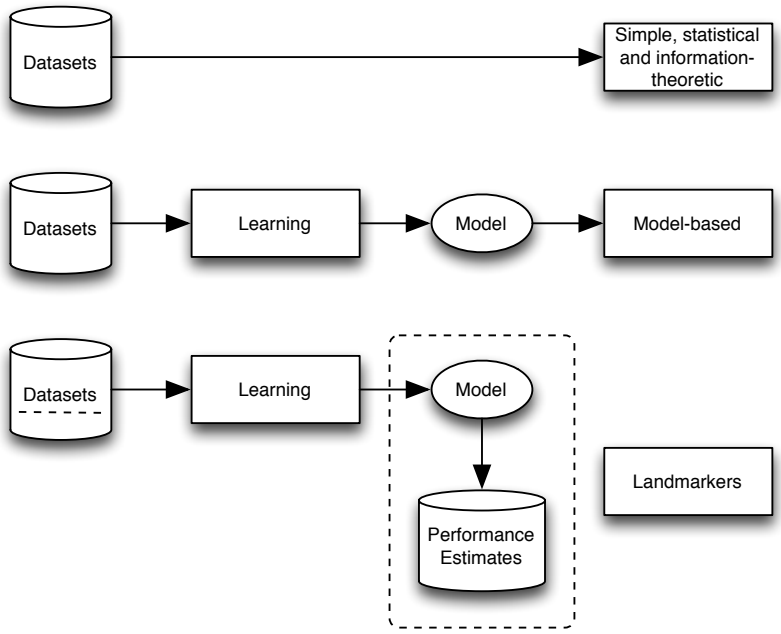

Figure 2.2: Metafeatures taxonomy. Source: Brazdil *et al.* [2009].

Brazdil *et al.* [2009] defined three fundamental issues that every metafeature should accomplish:

- *Discriminative power.* The metafeatures need to contain information that distinguishes between the base-algorithms in terms of their performance.

- *Computational complexity.* The computation of the metafeatures should not be too demanding. If not, it may not compensate generating a MtL system if one can save resources by exploring all the hypotheses for a given learning problem. Pfahringer *et al.* [2000] suggested that the computational complexity of extracting metafeatures should be at most $O(n \log n)$.

- *Dimensionality.* Given that the number of meta-examples of a MtL problem is usually small, the number of metafeatures should not be too large or overfitting may occur. Kalousis and Hilario [2001] found evidence that feature selection can improve a MtL process which supports this claim.

Each example in a meta-dataset represents a learning problem. As in other learning task, MtL needs a satisfactory number of examples in order to induce a reliable model. The number of meta-examples is often seen as a problem for MtL [Brazdil *et al.*, 2009]. Soares [2009] proposed a framework that enables to tackle this problem by generating different versions of a dataset through changes in the role of features/target.

Concerning the development of a MtL system, the first decision that must be made is about the type of *metatarget*, in other words, the dependent variable of the meta-level learning process. This variable can take several forms depending on the main goal of the MtL system and the nature of the base-level task (i.e., classification, regression or other). The most simple form of metatarget is a classification scheme (binary or multi-class, depending on the number of algorithms) in which for a given dataset, the metamodel predicts the class that represents the algorithm with better performance from a set. The great disadvantage of this type of metatarget is if the metamodel fails its prediction, the costs can be very high.

Another type of metatarget, instead of a single recommendation, is the suggestion of a subset of algorithms. Given the algorithm with expected best performance, a heuristic measure can be defined to indicate the algorithms that also perform well in comparison with the best algorithm. Typically, these meta-

models are induced with rules or Inductive Logic Programming [Kalousis and Theoharis, 1999].

Predicting a subset of algorithms provides several recommendations for the user. However, they are not ordered. This can negatively influence the data mining process. Therefore, algorithm recommendation in form of rankings seems a good alternative. A MtL method that provides recommendations in the form of rankings is proposed in the paper by Brazdil *et al.* [2003]. The system includes an adaptation of the k-nearest neighbours algorithm that identifies algorithms which are expected to tie, providing a reduced ranking by only including one of them in the recommendation.

Finally, if one is interested in concrete value regarding the performance of an algorithm in a dataset and not the actual relative performance of a set of algorithms, the metatarget can be defined as estimates of performance. In this case, the MtL problem takes the form of a regression, one for each base-algorithm. Besides that this type of metafeature provides more detailed information to the user, it also allows to transform the output of the metamodels in one of the previous recommendations forms mentioned above.

2.6.2 Applications

[Lemke *et al.*, 2015] identified in the literature a multiple use of the term MtL for distinct concepts, specifically:

- *Ensemble methods and combination of base-learners.* Algorithms for combination of base-models, such as bagging, boosting or stacking, are often regarded as MtL.

- *Algorithm recommendation.* Probably the area for which has been devoted the largest amount of MtL research. Most systems attempt to learn a relationship between data characteristics and algorithm performance.

- *Dynamic bias selection.* In this area, MtL algorithms are mostly designed for bias management and detecting concept drifts, typically in data streams environments.

- *Learning to learn.* Also known as *inductive transfer*, the goal here is to

transfer knowledge across multiple related domains or tasks, and therefore accumulate experience that can be useful in future tasks.

- *Metalearning systems.* This area regards the development of systems that automate specific tasks of a data scientist or provide assistance in those tasks.

We follow this organization to give an overview of the state-of-the-art for the main applications in each topic.

Ensemble methods

We already mentioned algorithms such as bagging, boosting and stacking in the section 2.2 of this thesis, so we refer the reader to that section to avoid repetition.

Algorithm recommendation

As mentioned before, Rendell *et al.* [1987] and Rendell and Cho [1990] published the first papers in which the expression "meta-learning" is used in Machine Learning. In the former, they proposed the Variable Bias Management System (VBMS). Here, the problem of algorithm recommendation is studied for the first time and the need for methods that develop models with different biases is identified. However, the experiment is rather preliminary: only the execution time of the algorithms is considered, the type of metafeatures is very simple (for instance, number of examples) and the evaluation carried is insufficient. In the latter, the data characterization was more detailed and set roots for the MtL research in following years, boosted by the already mentioned European projects, StatLog and METAL.

In the StatLog project, besides great developments in the scope of data characterization [Gama and Brazdil, 1995], MtL was used to predict the applicability of learning algorithms to a given data set [Brazdil *et al.*, 1994]. By applicability understand the notion of assessing if the performance of one learning algorithm is significantly different from the best algorithm on the corresponding dataset. The METAL project allowed to develop the research on MtL focusing more on the problem of ranking recommendations of algorithms [Brazdil *et al.*, 2003].

More recently, Sun and Pfahringer [2013] addressed this problem by proposing the Pairwise meta-rules (a new high-level framework to generate metafeatures) and the ART forests algorithm (a ranking algorithm based on Random Forests).

Kalousis *et al.* [2004] published a very interesting paper in which the authors looked for similarities between algorithms by means of error correlation, and similarities between datasets based on patterns of error correlation and relative performance of algorithms. Their main goal was not predictive performance at the meta-level, but gain understandable insights.

Also in the scope of classification, Ali and Smith [2006] used 112 datasets from UCI to infer a decision tree C4.5, producing rules for each algorithm with average accuracy of 10-fold cross-validation testing exceeding 80% in predicting the best algorithm. In majority, the metafeatures of the study were simple, statistical and information-theoretic.

Regarding the problem of algorithm parameter recommendation, Soares *et al.* [2004] proposed a MtL method to recommend values to set the width of the Gaussian kernel in a SVM. Later [Soares and Brazdil, 2006], they extended their work by showing that significant improvements could be achieved in the same problem after integrating metafeatures based on the kernel matrix. More recently, this problem of parameter recommendation for SVM has been extended by combining a MtL method with metaheuristics [Gomes *et al.*, 2012].

A more in-depth state-of-the-art on this topic can be found in the related work of chapter 3.

Dynamic bias selection

In this topic, MtL systems are developed with the purpose of bias management or detection of concept drifts, making the systems particularly suitable for data streams scenarios. We found some overlap between the methods presented in this section with the ones already presented in section 2.5. We decided to present here the systems that somehow include a *learning stage at the meta-level*.

Gama and Kosina [2013] proposed a MtL framework that reuses previously learned models on recurring contexts. Their approach differs from the typical MtL approaches in the sense that it uses the base-level features to train the meta-model. By contrast, Rossi *et al.* [2014] reported a system for periodic

algorithm selection that uses data characteristics to induce the meta-model (all metafeatures are simple, statistical or information-theoretic based). van Rijn *et al.* [2014] proposed a similar system but their approach uses a set of metafeatures with a large number of landmarkers.

Apart the data streams scenario, other papers have proposed MtL systems for dynamic bias selection. One of the most widely known applications of MtL to EL is due to Todorovski and Džeroski [2003], with the Meta-Decision Trees (MDT). The authors proposed an algorithm for learning a decision tree based on C4.5 that instead of making a prediction, the leaves of the tree specify which classifier should be used to obtain a prediction. Their study comprised 21 classification datasets and 5 base-level classifiers, namely, two algorithms for learning decision trees, a rule learning algorithm, a nearest-neighbour algorithm and a naive Bayes algorithm.

They test MDTs using two types of attributes: ordinary base-level attributes and metafeatures. The later reflect the certainty and confidence of the predictions and can be considered metafeatures. The simplicity of the approach allowed MDTs that are easy to interpret and useful metaknowledge can be extracted from the trees inspection. The metafeatures used were the *highest class probability*; *the entropy of the class probability distribution* and *the fraction of the training examples* used by the classifier to estimate the class distribution for a given example.

Recently, Cruz *et al.* [2015] proposed META-DES, a MtL method for dynamic selection of classifiers. It uses a meta-model to decide if each classifier of the ensemble is competent to classify a new instance. The meta-model is trained with the following metafeatures: 1) neighbours' hard classification - if the classifier is able to correctly predict the nearest neighbours; 2) posterior probabilities of the predictions made by the classifier in the nearest neighbours; 3) overall local accuracy; 4) output profile of the classifier in the decision space and 5) perpendicular distance between the test instance and the decision boundary. The selected classifiers are then combined using majority voting. Results show that META-DES is superior to other dynamic and static techniques such as OLA, LCA, KNORA-E, KNORA-U, among others.

More recently, Cerqueira *et al.* [2017] extended the original Arbitrating

framework [Ortega *et al.*, 2001] for time series forecasting tasks. Their method consists of an ensemble of heterogeneous models, arbitrated by a MtL model. The approach makes use of the base-level features to predict the error of each model in the ensemble. Then, the meta-prediction is used to weight and select each model.

This topic is highly related with the contributions of chapter 6 regarding the dynamic combination of ensembles, so we refer the reader to the related work of that chapter for more details on this.

Learning to learn

The ability to learn across different tasks and, thus, accumulate experience that enables to improve performance through time, is one of the key elements of human learning. Therefore, some ML researchers argue that such component is also a key element in building artificial intelligence [Lake *et al.*, 2016].

One of the first papers on learning to learn are by Schmidhuber [1992, 1993], in which the author proposed a neural network that is able to learn how to modify its own weights in order to improve its generalization ability. This line of thought ignited several papers in the past few years. In Andrychowicz *et al.* [2016], the output of MtL is a trained recurrent neural network (LSTM). The meta-learner is subsequently used as an optimization algorithm to learn other models from data; the optimization problem in the target learning task is cast as a learning problem itself. Ravi and Larochelle [2017] extended the work of Andrychowicz *et al.* [2016], by using a similar LSTM meta-learner in a few-shot classification scenario. In their approach, the traditional learner was a Convolutional Neural Network classifier. In summary, *learning to learn* research is presenting results (particularly within the deep learning community) that indicate that we should not only learn representations from data but also the optimizers that lead the learning process.

Metalearning systems

The Knowledge Discovery Process (KDD) includes several stages Fayyad *et al.* [1996]. One of the goals of MtL research is to automate the KDD process or at least some of its stages (most of the work usually focus on the modelling stage).

Some systems have been proposed with that aim. We provide an in-depth state-of-the-art for this topic in the related work of chapter 4.

More recently, some papers were published with the goal of automating one of the most challenging and time-consuming stages of the KDD process: feature engineering. The systems proposed take as input relational databases and use computational power to search for features with statistical relevance to a specific target variable [Lam *et al.*, 2017; Katz *et al.*, 2016; Kanter and Veeramachaneni, 2015]. We believe that this will be an important topic for MtL in the near future.

Part II

Automated Machine Learning

Chapter 3

Systematic Generation of Metafeatures

3.1 Introduction

One of the important steps in a data mining workflow is to decide which learning algorithm should be used to train on a given dataset. Since the number of learning algorithms is growing steadily, this step is becoming more difficult and time consuming. In the age of big data and data streams, the need for automated tools that enable fast experimentation and deployment is critical [Serban *et al.*, 2013]. Such systems must reduce the amount of time for model development without significant loss of model performance when compared to the best learning algorithm that could possibly be used.

Metalearning (MtL) is one approach that can be used to address this need. Brazdil *et al.* [2009] defined MtL as *the study of principled methods that exploit meta-knowledge to obtain efficient models and solutions by adapting machine learning and data mining processes.*

The quality of the meta-knowledge obtained with MtL depends on the availability of useful metafeatures (i.e. data characteristics that reflect the behaviour of learning algorithms). Although there have been proposed many metafeatures of different types for a wide range of problems (e.g. statistics-based measures and landmarkers), most of those metafeatures are developed in an *ad hoc* way.

For instance, some papers report the use of the entropy function on the target variable in classification problems, i.e. the *class entropy* metafeature, but only a few use metafeatures based on the application of the same function to feature variables, i.e. *attribute entropy* [Brazdil *et al.*, 2003]. Very often, there is no justification for such options. We believe that the reason for this is the lack of a method to guide the development of metafeatures.

Our proposal is a framework that supports the simple generation of an extensive set of meta-features simply by indicating a function. Then, the framework establishes how to systematically generate metafeatures from all possible combinations of arguments and post-functions alternatives that are compatible with that given function. For instance, rather than using *class entropy* and *attribute entropy*, the developer of a MtL approach chooses the function entropy and the system generates all the possible meta-features based on entropy (including *class entropy* and *attribute entropy*). An implementation of the framework is available in open source.[1]

We tested the approach in 203 classification datasets from the OpenML platform for collaborative ML [Vanschoren *et al.*, 2014]. The goals of the experiments are threefold: 1) compare the systematic process of generating metafeatures to a non-systematic one; 2) compare systematic sets of metafeatures based on simple measures to the state-of-the-art metafeatures; and 3) generate novel and informative sets of metafeatures by applying the proposed framework with functions that have not yet been used for MtL. The results obtained on the latter show that commonly used functions such as Pearson's correlation can be replaced by alternative meta-functions that generate more informative metafeatures.

In this chapter, we identify the following contributions:

- a framework designed with the purpose of making the metafeature generation process less *ad hoc* and more systematic
- a detailed presentation of the framework focused on the user's perspective
- novel sets of metafeatures generated by applying the framework

[1] Python implementation available in: https://github.com/fhpinto/systematic-metafeatures.

- an extensive empirical evaluation of the framework with an in-depth discussion of results

This chapter is organized as follows. Section 3.2 describes the state-of-the-art in MtL regarding applications and metafeatures. In section 3.3 we present the framework that supports the systematic generation of metafeatures. Section 3.4 shows how the framework can be used to represent and understand metafeatures previously used in the literature. In section 3.5 we present the results of the MtL experiments. Section 3.6 presents a more in-depth discussion of the results with particular emphasis on the set of most informative metafeatures that we found in our experiments. Finally, section 3.7 presents the conclusions of the chapter and indicates some directions for future work.

3.2 Metalearning

Figure 3.1 illustrates a common MtL framework for algorithm recommendation. The process starts with a collection of datasets and learning algorithms. For each of those datasets, we extract metafeatures that describe their characteristics (*step A in Figure 3.1*). Then, the performance of each algorithm on every dataset is estimated (B). The metafeatures and the estimates of performance are stored as metadata. The process continues by applying a learning algorithm (a meta-learner), which induces a meta-model that is a function of the relationship between the metafeatures and the best algorithm (C). Given the metafeatures of a new dataset (D), this meta-model is used to recommend an algorithm for that dataset (E).

3.2.1 Types of metafeatures

As in any other Machine Learning (ML) task, the predictive ability of a meta-model depends on the availability of informative (meta)features. The literature often groups metafeatures into three types: 1) simple, statistical and information-theoretic 2) model-based and 3) landmarkers [Brazdil *et al.*, 2009]. In the first group we can find the *number of examples* of the dataset, *correlation between numerical features* or *class entropy*, to name a few. This kind of metafeatures has the potential to provide interesting meta-knowledge (i.e.

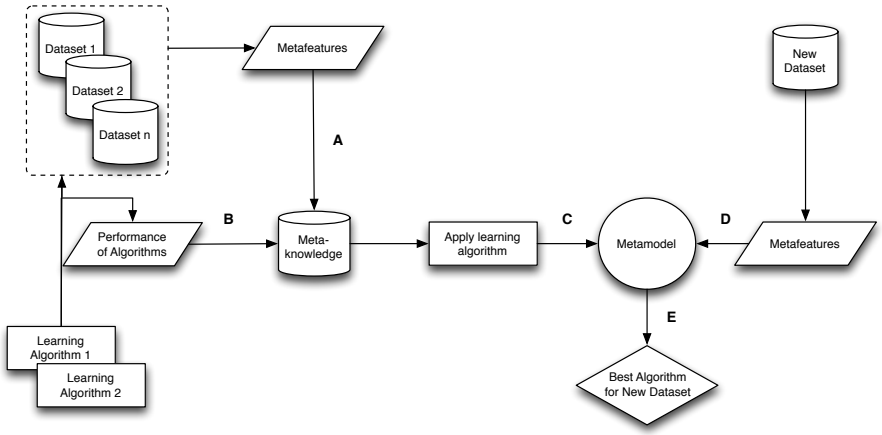

Figure 3.1: Metalearning framework for algorithm recommendation.

knowledge about the relation between the characteristics of datasets and the performance of the algorithms) because they typically represent properties that are familiar to ML experts [Brazdil *et al.*, 2003]. The model-based metafeatures capture some characteristic of a model generated by applying a learning algorithm to a dataset [Peng *et al.*, 2002b] (e.g., the *number of leaf nodes of a decision tree*). Finally, landmarkers are generated by obtaining a quick performance estimate of a naive learning algorithm in a particular dataset [Pfahringer *et al.*, 2000]. For instance, the predictive performance of a Decision Stump.

3.2.2 Domains of application

The main focus of MtL research has been on the problem of algorithm recommendation and most commonly applied to classification problems. Brazdil *et al.* [2003] provide recommendations in the form of rankings of learning algorithms. They used simple, statistical and information-theoretic metafeatures. Sun and Pfahringer [2013] extended the work of Brazdil *et al.* [2003] with two main contributions: the pairwise meta-rules (PMR), a higher-level type of metafeatures generated by comparing the performance of individual base learners in a one-against-one manner; and a new meta-learner for ranking algorithms. Their set of metafeatures consisted mostly of landmarkers, besides the already mentioned PMR. Other relevant papers that address the same issue are Kalousis and Hilario [2001] and Castiello *et al.* [2005], and both of them report the use of simple,

statistical and information-theoretic metafeatures.

MtL has also been applied for other purposes: time series forecasting [Prudêncio and Ludermir, 2004; Wang *et al.*, 2009; Lemke and Gabrys, 2010], parameter tuning [Soares *et al.*, 2004; Ali and Smith-Miles, 2006; Reif *et al.*, 2012a], data streams [Gama and Kosina, 2013; Rossi *et al.*, 2014; van Rijn *et al.*, 2014], clustering [de Souto *et al.*, 2008], among others [Brazdil *et al.*, 2009]. This lead to a large set of metafeatures proposed in the literature for very different problems.

One of the first attempts to use MtL to select the best method for time series forecasting was carried by Prudêncio and Ludermir [2004]. They used a set of simple and statistical metafeatures in their system. Wang *et al.* [2009] addressed the same problem with a descriptive MtL approach and a similar set of metafeatures. Lemke and Gabrys [2010] extended the set of metafeatures for this problem by grouping them into four categories: frequency domain, general statistics, autocorrelations and diversity. According to the categorization described earlier, this set of characteristics is considered to be simple and statistical metafeatures. More recently, Cerqueira *et al.* [2017] extended the original Arbitrating framework [Ortega *et al.*, 2001] for time series forecasting problems. Their approach consists of an ensemble of heterogeneous forecasters, arbitrated by a metalearning model. The method does not make use of metafeatures, relying instead on the base-level features to predict the error of each model of the ensemble.

MtL has also been used to tune parameters of learning algorithms. Soares *et al.* [2004] proposed a method that uses mainly simple, statistical and model-based metafeatures to predict the width of the Gaussian kernel in Support Vector Regression. Ali and Smith-Miles [2006] published a MtL method to automatically select the kernel of a Support Vector Machine in a classification scenario using a set of simple and statistical metafeatures. Reif *et al.* [2012a] used a MtL approach to provide good starting points for a genetic algorithm that optimizes the parameters of a Support Vector Machine and a Random Forests classifier. They used a set of simple, statistical and landmarker metafeatures.

Data stream mining can also benefit from MtL, especially in a context where the distribution underlying the observations may change over time. Gama and Kosina [2013] proposed a metalearning framework that reuses previously learned

models on recurring contexts. Their approach differs from the typical MtL approaches in the sense that it uses the base-level features to train the meta-model. By contrast, Rossi *et al.* [2014] reported a system for periodic algorithm selection that uses data characteristics to induce the meta-model (all metafeatures are simple, statistical and information-theoretic). van Rijn *et al.* [2014] proposed a similar system but their approach uses a set of metafeatures with an high percentage of landmarkers.

As mentioned before, one of the goals of MtL as a research field is to support the development of tools for automatic design of data mining workflows. Some attempts have been made with that goal, such as the work by Feurer *et al.* [2015]. They use a set of 38 simple, statistical and information-theoretic metafeatures for the initialization of their Bayesian optimization method.

3.2.3 Methodologies for metafeature design

All the MtL applications that we mentioned use different sets of metafeatures. It is common to find discrepancies between the use of a function such as entropy or mutual information to measure a specific element and not another. In some cases, these differences are forced by the nature of the problem and/or data. For instance, metafeatures that characterize the target feature in regression cannot be used directly in classification. However, we believe that it would be useful to decompose all these metafeatures into a common framework. Furthermore, such framework must also help the MtL user to systematically develop new metafeatures.

To the best of our knowledge, this work is the first attempt at the systematic design of general metafeatures. The approach that can be resembled as more similar to ours is the one proposed by Reif *et al.* [2012b]. However, they only focused on the post-processing phase of generating metafeatures and did not propose a framework that leads to a systematic approach to metafeatures generation.

An attempt on providing support to the generation of metafeatures for streaming data has been published but the approach is more concerned with the schematization of how the metafeatures were computed than actually providing a systematic approach for metafeatures generation [Rossi *et al.*, 2014,

2017]. Moreover, it is specific for data streams.

Landmarkers can also be regarded as a systematic approach to metafeature development since they are metafeatures that are generated based on base-level algorithms [Pfahringer *et al.*, 2000]. However, they are very specific to supervised learning problems and can be quite expensive computationally [Feurer *et al.*, 2015].

3.3 Systematic Generation Of Metafeatures

Our literature review on metafeatures shows that the sets of metafeatures used by researchers for MtL experiments is, most often than not, *ad hoc*. By this, we mean that there is no methodology for guiding the design of metafeatures that justifies their inclusion in the MtL experiments. Quite often some measures (such as Pearson's correlation or entropy) are used to capture some specific information of the learning process, while other metafeatures based on the same measures that might be equally important are left out. Table 3.1 illustrates this issue. For instance, entropy is commonly applied to the target variable of the datasets but not to the feature variables [Brazdil *et al.*, 2003]. This leads to sets of metafeatures that we consider *incomplete* in the sense that a *function* (such as entropy) has not been applied to all its possible arguments available for the problem. Therefore, we propose a framework to systematize the development of metafeatures for MtL problems. Our framework assumes that a metafeature can be represented by three elements (Figure 3.2): an argument tuple, a meta-function and a post-function.

Table 3.1: Some examples of metafeatures that have been used for MtL.

Metafeature	Argument	Meta-Function	Post-Function	Reference
Class Entropy	(Cat. Target)	Entropy	Non-aggregated	Brazdil *et al.* [2003]
Avg. Mutual Info.	(Cat. Features)	M. Information	Average	Brazdil *et al.* [2003]
Avg. Skewness	(Num. Features)	Skewness	Average	Castiello *et al.* [2005]
Corr. Histogram	(Num. Features)	Correlation	Histogram bins	Kalousis and Hilario [2001]
Landmarker	(Target, Predictions)	Accuracy	Non-aggregated	Pfahringer *et al.* [2000]

In the following subsections we provide more details about the framework and how to deploy it. We start by introducing some basic concepts, then we

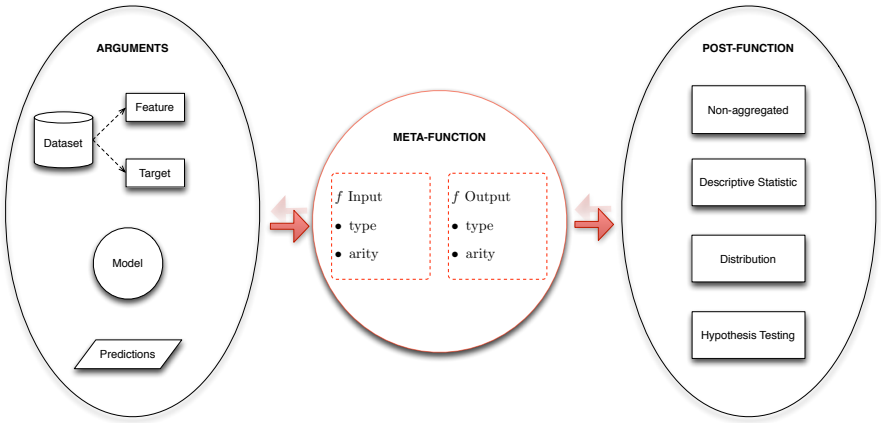

Figure 3.2: Framework for the systematic development of metafeatures.

discuss the algorithm and finally, we provide some detailed examples of its use.

3.3.1 Basic Concepts

For simplicity, we will focus on a supervised learning scenario to formalize the framework. Given a dataset D, composed of m features and a target vector Y, the purpose of a supervised learning problem is to find an hypothesis, which is a function $h : X \rightarrow Y$, where X is the input space, consisting of m features, and Y is the output space. A dataset D can also be described as an $n \times (m+1)$ matrix, containing n rows, representing the examples and $m+1$ columns, representing the features X_j, $1 \leq j \leq m$, and the target variable, Y. Both X_j and Y can be categorical or numerical variables.

$$D = \{X_j, Y\} \; \forall \; j = 1, ..., m \tag{3.1}$$

The characteristics of a dataset are obtained by applying a function to one or more elements of these types (for example, computing the entropy of a categorical target variable yields the metafeature *class entropy*). For convenience, we will represent these types using the following notation: `cat_feat` for categorical features, `num_feat` for numerical features, `cat_targ` for categorical targets and `num_targ` for numerical targets. Therefore, in other words, we can say that the metafeatures are computed using as input tuples of one or more elements of those types (repeating the example, *class entropy* is obtained by applying the

entropy function to an element of the type `cat_targ`). Therefore, the domain of types of elements that can be obtained from a dataset can be defined as:

$$T = \{\texttt{cat\_feat}, \texttt{num\_feat}, \texttt{cat\_targ}, \texttt{num\_targ}\} \tag{3.2}$$

Thus, we obtain a set A^D of elements from a given dataset D:

$$A^D = \{a_i\} \; \forall \; i = 1, ..., m+1 \tag{3.3}$$

and its types:

$$T^D = \{t_i^D : t_i^D \in T, \forall \; i = 1, ..., m+1\} \tag{3.4}$$

As shown in Figure 3.2, at the centre of the framework is a meta-function and it is the most important element of the three that make a metafeature. It is defined as in Definition 1.

Definition 1. *A meta-function f is a σ-ary function that takes as input an argument tuple, where the ith element is of one a set of types $t_i^f \subseteq T$ and $\sigma^f \in \mathbb{Z}^+$.*

It should be noted that elements of a dataset with different types can be used for the same argument. For instance, the entropy function is a meta-function with arity 1, which can be applied to arguments of types `cat_feat` or `cat_targ`.

Consequently, we can define T^f, the types of arguments that can be used with the meta-function f as:

$$T^f = \{T_i^f : T_i^f \subseteq T, \forall \; i = 1, ..., \sigma_f\} \tag{3.5}$$

Finally, the output of the meta-function may consist of multiple values. In this case, it may be necessary to post-process them. For instance, if the number of values computed by the meta-function depends on the number of variables of the dataset, they must be aggregated if a propositional approach to MtL is used (for instance, after computing the entropy of each categorical feature, it is necessary to aggregate this information with the mean to end up with the metafeature *average entropy of the categorical features*). Therefore, we define the set of post-functions as:

$$P = \{p_j\} \; \forall \; j = 1, ..., v \tag{3.6}$$

in which v is the number of post-functions. As mentioned before, a typical choice for post-function is the mean; other examples are median, maximum, minimum, standard deviation, variance or histogram bins [Kalousis and Theoharis, 1999].

3.3.2 Algorithm

Given the basic concepts introduced above, we now provide in Algorithm 2 the pseudo code that generates a set of metafeatures in a systematic manner.

$Calls_{\varnothing} = \{\{\}\}$
for $i \leftarrow 1$ **to** σ^f **do**
  $Calls_i = \varnothing$
  for $j \leftarrow 1$ **to** $|T^D|$ **do**
    for $e \leftarrow 1$ **to** $|Calls_{i-1}|$ **do**
      if $t_j^D \in t_i^f$ **then**
        $Calls_i = Calls_i \cup \{Calls_{i-1,e} \cup a_j\}$
      end
    end
  end
end
$Metafeatures_{\varnothing} = \{\}$
for $i \leftarrow 1$ **to** $|Calls_{\sigma^f}|$ **do**
  for $j \leftarrow 1$ **to** $|P|$ **do**
    $Metafeatures_{i,j} = p_j(do.call(f, Calls_{\sigma^f,i}))$
  end
end
return $Metafeatures$

Algorithm 2: Pseudo code for systematic generation of metafeatures.

Algorithm 2 generates the tuples obtained with all possible combinations of arguments obtained from a dataset D that are of the same type as the corresponding arguments to the function f. It then makes a call to f using each one of those tuples of arguments. Finally, it applies all possible post-function to the results of those calls.

To apply the algorithm, the user must initially identify the elements of the

framework as we described in the previous subsection, including, of course, the selection of a meta-function. Once this is accomplished, Algorithm 2 enables the generation of a set of metafeatures.

3.3.3 Example

For example purposes, consider a dataset in which $m = 3$. Therefore, it has three features: X_1, X_2 and X_3. Two of these features, X_2 and X_3, are categorical features and, therefore, of type `cat_feat`. The remaining one, X_1, is a numerical feature (`num_feat`). Since the dataset represents a classification problem, Y is therefore a categorical target vector (`cat_targ`).

To characterize this dataset, we could use some metafeatures that have proven to be informative. Table 3.1 shows some examples of those metafeatures. This is the *ad hoc* approach that is usually followed and has the drawbacks discussed earlier. To illustrate how the proposed framework addresses these drawbacks, we now use it to generate metafeatures for the dataset described above.

For instance, entropy is a meta-function with $\sigma^{entropy} = 1$, that takes categorical variables as input. Therefore, to the toy dataset described above, the meta-function arguments would be (X_2), (X_3), and (Y). On the other hand, if the meta-function was mutual information (which, again, takes categorical variables as input but $\sigma^{minfo} = 2$), the arguments would be all the combinations of size σ^{minfo} that can be generated with Y, X_2 and X_3. This process is illustrated in Figure 3.3.

Examples of post-functions commonly applied to the output of metafeatures are the average, minimum, maximum, standard deviation and variance. Following the same scenario described above, if the selected meta-function is mutual information, it can be applied to Y, X_2 and X_3. Therefore, following Algorithm 2, this generates three argument tuples: $\{(X_2, X_3)$, $((X_2, Y)$,$(X_3, Y))\}$. Assuming that the five post-functions mentioned above are available, this generates six metafeatures, composed by the following elements (*argument.meta-function.post-function*):

- (X_2, X_3).*MutualInformation.None*

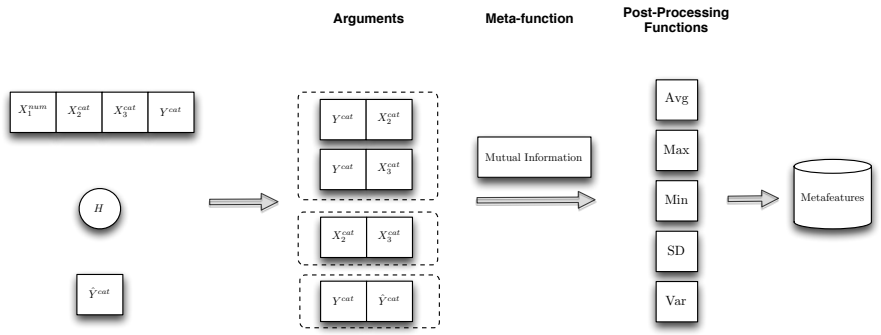

Figure 3.3: This schema illustrates the application of the framework in the example learning scenario provided in the text.

- $((X_2, Y),(X_3, Y)).MutualInformation.Average$
- $((X_2, Y),(X_3, Y)).MutualInformation.Maximum$
- $((X_2, Y),(X_3, Y)).MutualInformation.Minimum$
- $((X_2, Y),(X_3, Y)).MutualInformation.SD$
- $((X_2, Y),(X_3, Y)).MutualInformation.Var$

In here we focused on supervised learning problems. Furthermore, we have also focused on the generation of metafeatures from the features and the target. However, the framework is applicable to other types of elements, i.e. T can be extended, as Algorithm 22 is not dependent on T. An example of this is the work of Cunha *et al.* [2017], in which the framework is applied to a recommender systems learning scenario.

3.4 Fitting Common Metafeatures in the Framework

A first test to the validity of the proposed framework is to check if existing metafeatures could be the result of its use. We use examples from three types of metafeatures: simple, statistical and information-theoretic; model-based and landmarkers.

Figure 3.4 illustrates the representation in detail of five metafeatures using our framework. Furthermore, it makes it easier to find commonalities between metafeatures. For instance, the *absolute mean correlation between numerical features* is very similar to the *correlation between numerical features* (used in data streams applications as in Rossi *et al.* [2014]) except for the post-function. In latter it is feasible and potentially more informative to not aggregate the correlation values, because the recommendations are made for different samples of the same dataset (different periods of a single data stream), which, thus, have the same structure.

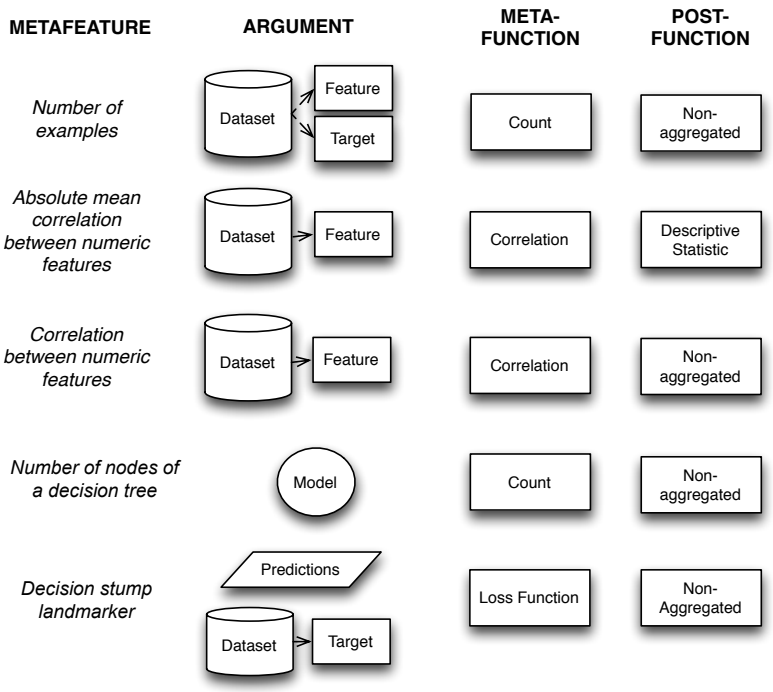

Figure 3.4: Metafeatures represented using our framework.

Still regarding Figure 3.4, the representation of the two last metafeatures shows that is possible to use the framework for more complex metafeatures. The *number of nodes of a decision tree* is an example of a model-based metafeature. The argument is the decision tree model, the meta-function is count and the post-function is non-aggregated. Peng *et al.* [2002b] propose several model-based metafeatures (for decision trees models) of this kind. Finally, we also

show an example of a landmarker. The *decision stump landmarker*, uses an argument tuple of predictions and the target variable. The meta-function here could be any type of loss function, such as accuracy.

A key aspect of our framework regards the completeness of the meta-function input, namely, the arguments. Given a meta-function, it is important to ensure that all possible metafeatures are generated. The input of the meta-function constrains the type of data that it can process. Therefore, by making sure that it is applied to all the possible sets of arguments, we ensure that the metafeatures created will represent all information that it is possible to collect using the selected meta-function.

An important advantage of our framework is that we only have to choose the meta-function, rather than designing all the metafeatures. A meta-function f is selected according to its relevance for the MtL problem. Although we acknowledge that this choice may be based on an *ad hoc* decision, the interest of a meta-function for a MtL problem can often be easily justified. For example, entropy is a concept used in several ML algorithms, including decision trees [Quinlan, 1986]. Therefore, metafeatures based on this function are expected to be useful to better understand the behaviour of those algorithms. Furthermore, this decision is made at a more abstract level than the typical design of metafeatures and is, thus, easier. For instance, it is indisputable that the concept of entropy is important for learning decision trees and that it is likely that some of the metafeatures that can be based on this function contain useful information on the behaviour of tree learning algorithms. On the other hand, a claim that *class entropy* is useful and *attribute entropy* is not, or vice-versa, would be harder to justify. Finally, given the choice of a meta-function, the framework generates metafeatures that characterize the arguments for which type is compatible. This makes sure that the metafeatures based on f that contain useful information, if any, will be generated.

The number of meta-functions that can be used in this framework is likely infinite. However, to aid the user in the process of using the framework to generate metafeatures, we refer a list of meta-functions that are often used as the basis for metafeatures: entropy, mutual information, Pearson's correlation, Spearman's correlation, skewness or kurtosis.

These are meta-functions that already have been used for MtL in a non-systematic way. Nevertheless, other meta-functions can be used which might generate more informative sets of metafeatures, such as the ones that we explore in section 3.5.6.

Recalling again Figure 3.2, the final part of a metafeature is the post-function, that can be categorized into four groups: *non-aggregated*, *descriptive statistic*, *distribution* and *hypothesis testing*.

The *non-aggregated* alternative uses the meta-function output in its raw state directly as metafeature(s). In some MtL problems it might be useful to not aggregate the information. This is particularly frequent in MtL applications such as time series or data streams where the datasets have the same morphology [Rossi *et al.*, 2014] or when the MtL algorithm is relational [Getoor and Mihalkova, 2011]. For instance, instead of computing the mean of the correlation between pairs of numerical features, one could use the correlation between all pairs of numerical features. It can also be the case that the output of the meta-function does not need aggregation and, therefore, the non-aggregated post-processing function is applied. The *descriptive statistic* case is perhaps the most common approach to aggregate information and generate metafeatures. This can be accomplished by using functions such as the *mean*, *maximum*, *minimum*, *standard deviation*, *mode*, etc. However, such aggregation can cause loss of valuable information. The *distribution* type of post-processing functions capture a representation of the output provided by the meta-function by characterizing its distribution. This finer-grained aggregation can be achieved through the use of *histograms* with a fixed number of bins [Kalousis and Theoharis, 1999]. In this case, each bin is used as metafeature, providing a description of the distribution of the output of the meta-function. Finally, in the *hypothesis testing* type of post-processing functions, the output provided by a meta-function f is used to test an assumption. For instance, it can test whether the values follow a normal distribution. The output of this test (such as the p-value) is used as metafeature.

Regarding post-functions, we believe that it is impossible to ensure completeness (i.e. all possible post-functions are applied). On the contrary, we believe that it is important that the framework is open in this regard, enabling

new, possibly more informative post-functions are investigated in the future. We identified four categories that are inspired by the functions used in the metafeatures proposed in the literature. However, the number of possible alternatives within each category is very high and eventually infinite. Furthermore, since the raw output of the meta-function is available, then any function that is compatible with that output can be applied. The openness of the post-function part ensures that the framework is not limited to the set of functions that have been used so far.

3.5 Experiments

The experiments that we present in this section aim to answer three questions:

1. Given a function that is used as the basis for metafeatures in typical metalearning approaches, is the proposed framework able to develop sets of systematic metafeatures that are consistently more informative than non-systematic sets containing these typical metafeatures?

2. Is the set of systematically generated metafeatures more informative than the state-of-the-art metafeatures?

3. Can we use the framework to easily generate novel and informative sets of metafeatures that have not been used previously in other MtL approaches?

For the first question, we carried out a set of experiments with the goal of providing a proof of concept of our framework. By testing whether the systematic generation of metafeatures, with functions that are used in common metafeatures, leads to better results than with the original set of a non-systematic set of metafeatures, we show that our framework can be useful and help MtL users to avoid an *ad hoc* selection. For the second question, we executed experiments in which we compare the set of metafeatures generated by our framework with the state-of-the-art sets, such as the one provided by the OpenML platform [Vanschoren *et al.*, 2014] and the pairwise meta-rules [Sun and Pfahringer, 2013]. Finally, in the third set of experiments, we selected two meta-functions that, to the best of our knowledge, have never been used in the context MtL. We compare the sets generated by those meta-functions against the sets generated by

common meta-functions such as Pearson's correlation and mutual information.

For all experiments we followed a standard MtL approach of selecting the best algorithm in a set for classification problems. Figure 3.1 describes the approach.

3.5.1 Base-level experimental setup

All the experiments were executed on 203 classification datasets extracted from the OpenML platform [Vanschoren *et al.*, 2014]. This set of datasets results from filtering all the classification datasets available in the platform with the following rules: 1) less than 5000 and more than 300 examples 2) less than 1000 features. These filters were applied in order to speed up the experiments and avoid datasets that were too small or too large for our time and computational constraints. In terms of pre-processing of the datasets, we deleted the features with more than 15% missing values and/or near zero variance.

Six classification algorithms were tested as base learners: NaiveBayes, k-NN, C5.0, CART, SVM (with RBF kernel) and Random Forest. The estimates of algorithm performance were obtained using 10-fold cross validation and Cohen's kappa as error measure [Cohen, 1960]. The *R* package *caret* was used for parameter tuning of the learning algorithms [Kuhn, 2008]. The estimates of performance showed that NaiveBayes was better in 17 datasets, k-NN in 27, C5.0 in 30, CART in 43, SVM in 40 and Random Forest in 46.

3.5.2 Meta-level experimental setup

Since our preliminary experiments showed that Random Forests has a good performance at the meta-level, we chose it as the learning algorithm used to generate the meta-model. Several papers in the literature have reported that Random Forests is a robust option for modelling data at the meta-level [Hutter *et al.*, 2011; Sun and Pfahringer, 2013] and our results are in accordance with that.

Again, the estimates of performance of the meta-learner were obtained using 10-fold cross validation (averaged over 30 repetitions) and the error measure is accuracy (i.e. whether the meta-model was able to select the best algorithm for a dataset or not). As baseline, we use a very simple model that predicts

for all datasets that the best algorithm is Random Forest, that is, in fact, the majority class in our meta-dataset. Moreover, Random Forests is an algorithm that has shown state-of-the-art performance in several datasets across different domains [Fernández-Delgado *et al.*, 2014].

For statistical validation we used the methodology recommended by Demšar [2006]: Friedman rank test with Nemenyi test for post-hoc multiple comparisons.

3.5.3 Systematic sets of metafeatures

The sets of metafeatures were generated using the following types of elements:

- `num_feat`
- `cat_feat`
- `cat_target`
- `cat_pred` (from five landmarkers - naive bayes, decision tree with depth 1, 2 and 3, and majority class)

Given the types of elements available, we generated four sets of systematic metafeatures by selecting three meta-functions:

- entropy, which is an 1-ary function that takes as input elements of type `cat_ feat`, `cat_target` and `cat_pred`;
- mutual information, which is a 2-ary function that takes as input elements of type `cat_ feat`, `cat_target` and `cat_pred`;
- Pearson's correlation, which is a 2-ary function that takes as input elements of type `num_feat`;

Each meta-function generated 21, 35 and 15 metafeatures, respectively. A fourth set, *All*, gathers the metafeatures from the three previous sets (71 in total).

As post-functions, we used average, standard deviation, variance, minimum, maximum and histogram bins.

The choice of the meta-functions was not random. Entropy and mutual information are information theory concepts used in some machine learning algorithms, particularly decision trees (and derivatives). Finally, Pearson's correlation coefficient measures the linear dependence between two numeric variables. Therefore, it is expected that this function can capture redundancy between *numerical features*.

One of the challenges of using our framework to generate systematic metafeatures is the large number of meta-level variables that are generated, which may lead to results that are affected by the curse of dimensionality. Since MtL applications are often based on a small number of meta-examples (i.e. base-level datasets), it increases the probability of obtaining spurious (meta-)patterns. So, we rely on two feature selection algorithms to tackle this problem: ReliefF [Robnik-Šikonja and Kononenko, 2003] and correlation feature selection (CFS) [Hall, 1999].

3.5.4 Systematic vs non-systematic

The set of experiments presented in Table 3.2 aims to answer the first research question that we stated in the beginning of this section. For each meta-function selected, we generated a set of metafeatures using the systematic approach that we propose.

For instance, using entropy as meta-function, the systematic approach enables to generate the following metafeatures: *categorical target* entropy (1 metafeature), average, maximum, minimum, standard deviation and variance of the *categorical features* entropy (5 metafeatures), histogram bins aggregation of the *categorical features* entropy (10 metafeatures), and entropy of the set of *categorical predictions* made by each landmarker (5 metafeatures). In total, taking entropy as meta-function, the systematic approach enables to generate a set of 21 metafeatures. We name the systematically generated sets as *Syst* and we compare them with sets that we generate with a non-systematic approach.

Again, if we take entropy as example, the respective non-systematic *NonSyst* set is composed by the average entropy of the *categorical features* and five landmarkers metafeatures (that use accuracy as error measure). The same approach is applied to generate the non-systematic sets with the other meta-functions. For

mutual information, we add to the five landmarkers the average mutual information between the *categorical target* and the *categorical features* and the average mutual information between *categorical features*; for Pearson's correlation, we just add the average correlation between *numerical features*. Therefore, we are able to evaluate if a systematic approach to metafeatures generation yields gains in terms of predictive performance at the meta-level.

Table 3.2: Results comparing the systematic metafeatures generated with meta-functions entropy, mutual information and correlation in comparison with the non-systematic. $SystCFS$ and $SystReliefF$ represent the systematic sets after feature selection with the methods CFS and ReliefF, respectively. The baseline achieves an average 22.65% accuracy on all experiments. The standard deviation of the values is between parentheses.

	Meta-function			
	$Entropy$	$M.Info$	$Correlation$	All
$NonSyst$	29.45% (1.94)	30.95% (1.53)	37.29% (1.45)	40.59% (1.45)
$Syst$	33.75% (1.79)	33.12% (1.41)	44.17% (1.72)	47.18% (1.34)
$SystCFS$	30.51% (2.00)	35.42% (1.70)	43.75% (1.86)	45.87% (1.51)
$SystReliefF$	30.91% (1.99)	33.97% (2.03)	42.75% (1.71)	45.05% (2.00)

Table 3.2 shows the results of the experiments. Generally, the sets of metafeatures generated with our framework present a superior performance. This result is consistent across all meta-functions, including in the *All* set. It is noticeable that the difference between *NonSyst* and *Syst* is larger in the case in which the meta-function is Pearson's correlation. We present a deeper discussion on this result in subsection 3.5.6.

Regarding the use of feature selection methods to reduce the dimensionality of the meta-dataset, the experiments show systematically that it does not improve the performance of the meta-model. In fact, in most cases, the accuracy decreases, both for CFS and ReliefF. Interestingly, even in the experiments with the *All* set (in which the number of metafeatures increases to 71), the feature selection algorithms do not seem to improve the results.

Moreover, all meta-models generated, whether with the *Syst* set or the *NonSyst* set, show a superior performance than the baseline (the majority class, which in the case of our meta-dataset, is always choosing the Random Forests algorithm). This is indicative of the usefulness of a MtL system for assisting the selection of a learning algorithm for a given dataset.

Figures 3.5, 3.6, 3.7 and 3.8 show the critical difference (CD) diagrams for each set of metafeatures and meta-learning algorithm. We set $\alpha = 0.05$ for all experiments. Generally, the CD diagrams confirm the results already stated in Table 3.2. The *Syst* set is always better than the *NonSyst* set and the baseline, with the exception of the experiments carried out with mutual information. In that particular CD diagram we can verify that the *Syst* set is not significantly better than the *NonSyst* set. However, the feature selection algorithms do improve the results obtained with *Syst*, leading to *SystCFS* and *SystReliefF* being better than *NonSyst*.

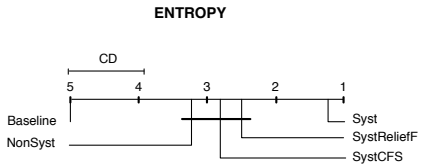

Figure 3.5: Critical Difference Diagrams for the entropy meta-function.

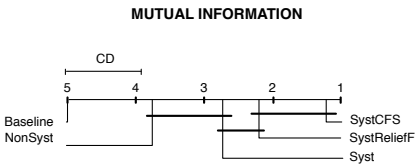

Figure 3.6: Critical Difference Diagrams for the mutual information meta-function.

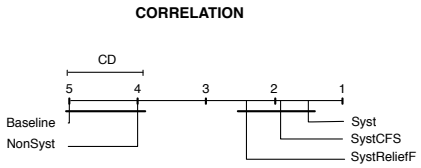

Figure 3.7: Critical Difference Diagrams for the correlation meta-function.

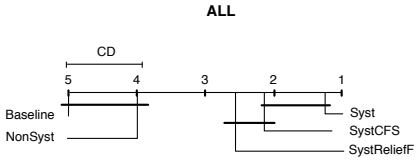

Figure 3.8: Critical Difference Diagrams for the three meta-functions combined.

3.5.5 Systematic vs state-of-the-art

In this subsection, we compare the sets of metafeatures generated by our framework against the most recent developments in terms of metafeatures design. We compare our approach of generating metafeatures with two sets of metafeatures with good results reported in the literature: the set proposed by Brazdil *et al.* [2003] and five landmarkers as introduced by Pfahringer *et al.* [2000]; and the set available in the OpenML platform [Vanschoren *et al.*, 2014] to describe the datasets that we collected.

Brazdil *et al.* [2003] proposed the following set of metafeatures: *number of examples*, *proportion of categorical features*, *proportion of missing values*, *proportion of features with outliers*, *class entropy*, *average mutual information of target and features* and *canonical correlation of the most discriminating single linear combination of numerical features and the target distribution.* To this set of simple, statistical and information-theoretic metafeatures we also added five landmarkers [Pfahringer *et al.*, 2000]: three *decision trees with depth 1, 2 and 3*, a *Naive Bayes* and a majority class predictor. For the purpose of this empirical study we named this set of 12 metafeatures as *SIL* (*Statistical, Information-theoretic and Landmarkers*).

The OpenML platform provides a set of 106 metafeatures that characterize the datasets stored in the database. This set is composed by 44 landmarkers and 62 simple, statistical and information-theoretic metafeatures. For the purpose of this empirical study we named this set as *OpenML*.

Recently, Sun and Pfahringer [2013] proposed pairwise meta-rules (PMR), a metafeature generation method based on rules that compares the performance of individual base learners in a one-against-one manner. For each pair of algorithms (since we test 6 base-learners, we have 15 pairwise comparisons) the method generates on average two PMR for each pairwise comparison. So, in our experimental setup, using PMR implies adding roughly 30 new metafeatures to the original set of metafeatures. So, we added PMR metafeatures both to *SIL* and *OpenML* sets, forming *SIL + PMR* and *OpenML + PMR*, respectively. For completeness, we also added PMR metafeatures to the PMR set, forming *Syst + PMR*.

Table 3.3 presents the results of the experiments. Comparing the sets of

metafeatures, it is noticeable that those generated using our framework obtain better predictive performance in comparison with the other sets. The difference is more noticeable comparing the *Syst* set with the *SIL* set. The differences are statistically significant as can be seen in the Critical Difference diagrams in Figure 3.9.

Table 3.3: Systematic sets of metafeatures in comparison with state-of-the-art metafeatures. The baseline achieves a 22.65% accuracy on all experiments. The standard deviation of the values is between parentheses.

	w/o PMR	*w/ PMR*
Syst	47.23% (1.68)	46.67% (2.03)
SIL	37.54% (1.85)	33.36% (1.65)
OpenML	44.29% (1.54)	44.36% (1.51)

Regarding the addition of the PMR, there is no significant gains in the predictive performance of the meta-models. In the case in which including PMR has better performance (*OpenML + PMR*), that difference is not statistically significant, as shown in Figure 3.9. We believe that this result is influenced mostly by the fact the we are not performing a ranking task at the meta-level, by comparison to the experimental setup that [Sun and Pfahringer, 2013] opted for. Since the pairwise meta-rules are generated by comparing algorithms in a one-against-one manner, it is expected that it yields better results for ranking tasks. However, our experimental setup focuses on algorithm selection as a classification task.

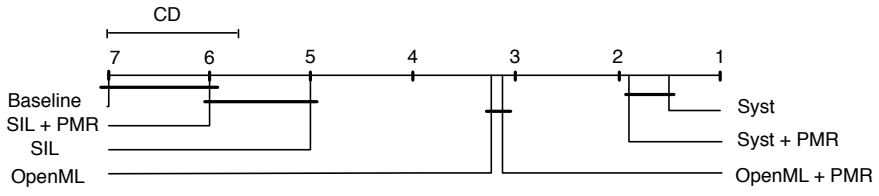

Figure 3.9: Critical Difference diagrams of systematic metafeatures Vs state-of-the-art.

3.5.6 Generating novel sets of systematic metafeatures

The framework enables the development of metalearning approaches without focusing on the detailed design of metafeatures but rather the identification of functions that potentially represent information about the behaviour of the learning algorithms considered.

Common metafeatures typically identify simple relations between the variables and represent information about one or at most two variables, so we looked for functions that could represent non-linear relations and the interactions between more than two variables:

- *Maximal Information Coefficient (MIC)*: a 2-ary function that measures the strength of the linear or non-linear relationship between two numeric variables. We will test the set of metafeatures generated by this meta-function against the one generated by Pearson's correlation, a meta-function that only measures linear relationships. We want to assess if a meta-function capable of measuring non-linear relationships between elements of type `num_feat` can generate a set of more informative metafeatures. Details on *MIC* can be found in the original paper by [Reshef *et al.*, 2011].

- *Interaction Information (InterInfo)*: a 3-ary function that is a generalization of mutual information for three discrete variables [2] [McGill, 1954]. Roughly, it measures the amount of information contained in a set of variables beyond the information that is contained in any subset of those variables. Our motivation is to assess if a more informative set of metafeatures can be generated with a meta-function that is able to measure the information in more than two `cat_feat`, as it happens with mutual information.

To the best of our knowledge, both *MIC* and *InterInfo* have never been used as part of a MtL approach.

One disadvantage of using a meta-function with an arity larger than 2, such as *InterInfo*, is the computational cost. Since we are computing the information in every possible triplet of `cat_feat`, the number of computations can grow exponentially for some datasets.

[2] Actually, the arity of this function can be any positive natural number.

Table 3.4 shows the results of the comparison between the set of metafeatures generated with Pearson's correlation and the one generated with *MIC*. The values show that the latter obtain a better performance. However, the CD diagrams that we provide of these experiments in Figure 3.10, show that the difference is not statistically significant.

Table 3.4: Comparison of the sets of metafeatures generated by Pearson's correlation and MIC. The baseline achieves a 40% accuracy on all experiments. The standard deviation of the values is between parentheses.

CORR vs MIC	
$SystCorr$	44.33% (1.19)
$SystMIC$	45.57% (1.43)

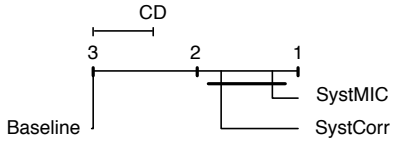

Figure 3.10: Critical Difference diagrams comparing the set of metafeatures generated by Pearson's correlation with the set generated by MIC.

Table 3.5 shows the results of the comparison between mutual information and interaction information. Besides the greater computational cost, there is no improvement in using Interaction Information instead of Mutual Information. This difference is statistically significant (as we can see in the CD diagrams provided in Figure 3.11).

The results obtained with *MIC* are very promising. This indicates that meta-functions with the ability to measure non-linear relationships can be very important for MtL. However, further research is needed since our statistical validation of the hypothesis was not favourable.

Table 3.5: Comparison of the sets of metafeatures generated by mutual information and interaction information.The baseline achieves a 22.65% accuracy on all experiments. The standard deviation of the values is between parentheses.

MUT. INFO. vs INTER. INFO.	
$SystMI$	31.78% (1.72)
$SystInterInfo$	27.29% (1.43)

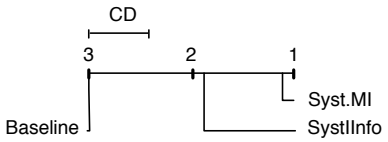

Figure 3.11: Critical Difference diagrams comparing the set of mefeatures generated by mutual information with the set generated by interaction information.

3.6 Discussion

In this section we provide a more in-depth discussion of the results obtained in the previous section. We focus particularly on the metafeatures that showed to be more informative for the meta-models that we developed. This aims to provide both a better understanding of the results obtained and also gain insights to guide future MtL experiments.

Table 3.6 shows the variable importance of six meta-models generated with the six different sets of metafeatures, generated using as meta-functions: entropy, mutual information, Pearson's correlation, *MIC*, interaction information and all (that includes all the metafeatures from the five sets that we mentioned). The meta-models were generated using all metadata collected from the 203 datasets. All meta-models were generated using Random Forests as learning algorithm and the variable importance was measured using the mean decrease in Gini index by removing the metafeature in question.

From the point of view of the meta-functions, it was interesting to see that a mixture of metafeatures generated from different meta-functions (that measure

Table 3.6: List of the top 3 metafeatures in terms of variable importance for Random Forests meta-models. For each set of features, we generated a meta-model using Random Forests as meta-learner using all the metadata collected from the 206 datasets. The variable importance is measured using the mean decrease in Gini index.

	Metafeature Importance		
	1st	2nd	3rd
Entropy	DTreeD3.Entropy.None	NBayes.Entropy.None	DTreeD1.Entropy.None
MInfo	NBayes.MInfo.None	ClassMajor.MInfo.None	DTreeD3.MInfo.None
PCorr	X.PCorr.Min	X.PCorr.Avg	X.PCorr.HistBin1
MIC	X.MIC.Min	X.MIC.Avg	X.MIC.HistBin1
InterInfo	DTreeD2.InterInfo.None	NBayes.InterInfo.None	DTreeD1.InterInfo.None
All	NBayes.MInfo.None	DTreeD3.Entropy.None	X.PCorr.Min

different aspects of the information available) pays off in terms of predictive performance (as seen in Section 3.5.4). The same result is visible in this analysis, since the top 3 metafeatures in the *All* set (that includes metafeatures from five meta-functions) has metafeatures from three different meta-functions, namely: mutual information, entropy and Pearson's correlation.

Based on Table 3.6, we observe:

1. Predictions from landmarkers are an important source of information to design informative metafeatures.

2. Using the minimum as post-processing function is a better option than using the average (which is very common in metafeatures design).

Regarding 1), the fact that landmarkers are informative is not a novelty, since this has been vastly reported in the literature. However, it is interesting to see that using a different meta-function (than lets say, accuracy) to measure information in predictions made by landmarkers models can be quite informative. For instance, the top 3 metafeatures in the *Entropy* set are landmarkers. Since entropy is a meta-function that only takes one argument as input, it is no possible to relate the predictions with the ground truth (only the entropy of the predictions is measure). This type of metafeature shows here to be informative, both in the *Entropy* set and *All* set.

Concerning observation 2), this is a more surprising result. It is very common

to use the average as a post-function for metafeatures (i.e., average feature entropy, average mutual information between target and features). However, what we verified systematically in our experiments is that aggregating the output of a meta-function with other post-processing functions such maximum, minimum or histogram bins, is more informative than averaging the values. We believe that this result is explained by the fact that the performance of an algorithm is probably better explained by the existence or not of a single, very informative feature than the average information in all. The minimum and maximum values of the meta-features may, thus, be more informative than the average value.

3.7 Conclusions and Future Work

This chapter presents a generic framework to develop metafeatures for MtL problems. We believe that this framework is important for the development of MtL approaches, because it allows the researcher to develop extensive sets of metafeatures by focusing on the choice of a simple function that extracts information from the data that is relevant for the behaviour of the algorithms.

The framework is structured in such a way that the systematic generation of metafeatures is triggered by the selection of a meta-function. Then, given the arguments and post-functions that are available, the framework outputs a set of metafeatures that are generated systematically. The process can be repeated with several meta-functions. The selection of the meta-function is crucial and it should be chosen intuitively according to the MtL application, in particular, the set of base level algorithms considered.

We used the framework to analyse several metafeatures proposed in the literature for a wide range of MtL scenarios. This process validated the framework by showing that it is consistent with several state-of-the-art metafeatures.

Our experiments address three questions: (1) are the systematic sets of metafeatures more informative than the non-systematic ones? (2) are the systematic sets generated with the framework better than the state-of-the-art? 3) can we use the framework to generate novel and informative sets of metafeatures with meta-functions that have not been previously used for MtL?

In the first set of experiments, we found that the systematic metafeatures

generated are consistently more informative than the non-systematic ones. This result confirms our hypothesis that a systematic approach could be more informative than an *ad hoc* selection of metafeatures.

In the second set of experiments, we found that the systematic sets are more informative than two state-of-the-art sets of metafeatures, with or without the addition of PMR.

Finally, in the third set of experiments, we used the framework to generate novel sets of metafeatures by using meta-functions that, to the best of our knowledge, have not yet been used in MtL approaches. Common metafeatures typically identify simple relations between the variables and represent information about one or at most two variables, so we looked for functions that could represent non-linear relations and the interactions between more than two variables. We selected MIC and interaction information that can be regarded as alternatives (or complements) to Pearson's correlation and mutual information, respectively. The results show that set of metafeatures generated by MIC enable an improvement over the one generated by Pearson's correlation. This is a relevant contribution since Pearson's correlation has been widely used for MtL. We could not verify the same improvement for interaction information. Furthermore, given the computational cost, the use of interaction information instead of mutual information is not advised.

As for future work, we plan to use this framework to systematically generate metafeatures for different MtL problems, such as hyper-parameter tuning of learning algorithms. Another interesting development would be the integration of the framework in a experiment database, such as OpenML [Vanschoren *et al.*, 2014].

Chapter 4

autoBagging: Ranking Bagging Workflows with Metalearning

4.1 Introduction

Ensemble learning (EL) has proven itself as one of the most powerful techniques in Machine Learning (ML), leading to state-of-the-art results across several domains [Fernández-Delgado *et al.*, 2014]. Methods such as bagging, boosting or Random Forests are considered some of the favourite algorithms among data science practitioners. However, getting the most out of these techniques still requires significant expertise and it is often a complex and time consuming task. Furthermore, since the number of ML applications is growing exponentially, there is a need for tools that boost the data scientist's productivity.

The research field that aims to answers these needs is Automated Machine Learning (*autoML*). It is a field that merges ideas and techniques from several ML and optimization topics, such as Bayesian optimization, metalearning (MtL) and algorithm selection. In the past few years, some breakthroughs made possible the development of technique and tools that aid data science practitioners, including non-experts, to efficiently create fine-tuned predictive models.

In this chapter we address the problem of how to automatically tune an EL algorithm, covering all components within it: generation (how to generate the models and how many), pruning (which technique should be used to prune the ensemble and how many models should be discarded) and integration (which model(s) should be selected and combined for each prediction). Particularly, we focus on the bagging algorithm [Breiman, 1996a], one of the most popular EL algorithms that is also one of building foundations of Random Forests. Specifically, our method is able to automatically tune four components of the algorithm: 1) the number of models that should be generated 2) the pruning method 3) how many models should be pruned and 4) which dynamic integration method should be used. For the remaining of this chapter, we call to a set of these four elements a bagging workflow.

Our proposal is *autoBagging*, a system that combines a learning to rank approach together with MtL to tackle the problem of automatically generate bagging workflows. Ranking is a common task in information retrieval. For instance, to answer the query of a user, a search engine ranks a plethora of documents according to their relevance. In this case, the query is replaced by a new dataset and *autoBagging* acts as a ranking engine.

Figure 4.1 shows an overall schema of the proposed system. We leverage the historical performance of each workflow in several datasets, where each dataset is characterised by a set of metafeatures. This metadata is then used to generate a metamodel, using a learning to rank approach. Given a new dataset, we are able to collect metafeatures from it and feed them to the metamodel. Finally, the metamodel outputs an ordered list of the workflows, specifically tuned to the characteristics of the new dataset.

We tested the approach on 140 classification datasets from the OpenML platform for collaborative ML [Vanschoren *et al.*, 2014] and 63 bagging workflows. We give details on these workflows in section 4.4. Results show that *autoBagging* has a very competitive performance against other state-of-the-art methods, such as *auto-sklearn* [Feurer *et al.*, 2015]. Furthermore, testing the top 5 workflows recommended by *autoBagging* guarantees an outcome that is not statistically different from the *oracle*, an ideal method that for each dataset always selects the best workflow.

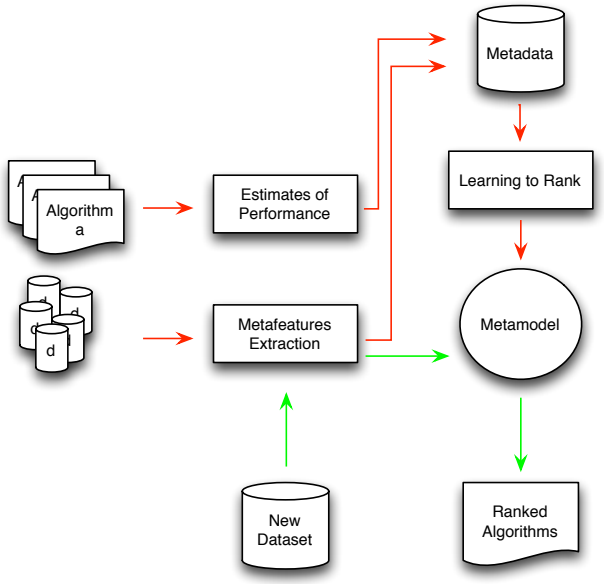

Figure 4.1: Learning to Rank with MtL. The red lines represent offline tasks and the green ones represent online ones.

This chapter is organized as follows. Section 4.2 describes the state-of-the-art regarding autoML and MtL, with particular emphasis to approaches more similar to ours. In section 4.3 we introduce the concept of bagging workflows and describe the components from each they are designed with. Section 4.4 formalizes the *autoBagging* method. Section 4.5 presents the experiments carried to evaluate our approach. Finally, section 4.6 concludes the chapter and sets directions for future work.

For the purpose of reproducibility and generalizability, *autoBagging* is publicly available as an R package. [1]

4.2 Related Work

In this section we provide a brief overview of systems designed for automatic recommendations or ranking of ML algorithms/workflows. After a careful analysis of the state-of-the-art, we split it into three categories: 1) systems that use only a MtL approach, without any kind of optimization component; 2) systems

[1] https://github.com/fhpinto/autoBagging

that make use of optimization procedures, such as bayesian optimization and 3) systems that leverage both MtL and optimization procedures. Finally, in the last sub-section, we discuss how some *autoML* systems recommend ensemble learning algorithms to the user and how our approach differs from previous ones regarding this feature.

4.2.1 Metalearning based

The first automated framework proposed to support machine learning processes was the Data Mining Advisor (DMA) [Giraud-Carrier, 2005]. The system used an instance-based learning approach to relate the performance of the learning algorithms with simple and statistical metafeatures computed from the datasets [Brazdil *et al.*, 2003].

This line of research was later on dominated by the characterization of datasets through landmarkers [Pfahringer *et al.*, 2000], such as learning curves [Leite and Brazdil, 2005] or pairwise meta-rules [Sun and Pfahringer, 2013].

4.2.2 Optimization based

Evaluating ML algorithms and/or ML workflows is typically very time consuming and computationally expensive. In practice, it is not feasible to evaluate all learning algorithms with 10-fold cross validation for a given dataset (particularly if the dataset is of high dimensionality) and choose the one that minimizes the error. Therefore, researchers have been developing search procedures and optimization algorithms that can in fact do this in reasonable time.

Bayesian optimization is the field within optimization that has had the most success carrying out these type of tasks. One of the algorithms that is responsible for this success is *SMAC*, an approach that constructs explicit regression models to describe the relationship between the target algorithm performance and the hyper-parameters [Hutter *et al.*, 2011]. The ability of *SMAC* to deal both categorical and continuous hyper-parameters is one of the reasons behind its success.

The development of bayesian optimization algorithms for algorithm configuration has led to the emergence of systems such as *Auto-WEKA*, that makes use

of *SMAC* and the machine learning library *WEKA* to automatically generate workflows for classification datasets [Thornton *et al.*, 2013].

More recently, the *hyperband* method was proposed as an alternative to bayesian optimization algorithms [Li *et al.*, 2016]. The method uses a pure-exploration algorithm for multi-armed bandits to exploit resources (such as the number of examples of the training set or number of epochs of a neural network) required by machine learning algorithms. The results that have been reported show better performance of the *hyperband* method over the bayesian optimization methods.

4.2.3 Optimization and metalearning

Some *autoML* systems combine bayesian optimization with MtL, particularly useful to act as a warm start for the optimization procedure. An example of such system is *auto-sklearn* [Feurer *et al.*, 2015]. Given a new dataset, the system starts by comparing the characteristics of that dataset with past performance of ML workflows on similar datasets (using a set of simple, statistical and information-theoretic metafeatures and k-NN). After this warm start, the optimization procedure is carried out by *SMAC*. Finally, the system also has the ability to form ensembles from models evaluated during the optimization.

4.2.4 Ensemble focused autoML

Some attempts have been made in creating *autoML* systems that are able to provide suggestions of ensembles. Again, and probably the most influential one, is *auto-sklearn*, as described above. Another proposal on this matter is made on Lacoste *et al.* [2014], where the authors optimize ensembles based on bootstrapping the validation sets to simulate multiple independent hyper-parameter optimization processes and combined the results with the agnostic Bayesian combination method.

One of the problems with the two approaches described is that the generation of the ensemble is rather *ad hoc*. That means, it does not take into account important properties that are known to affect the performance of ensembles. Specifically, complementarity between models and the overall diversity of the ensemble. We argue that the ensemble generation phase must take into

account these concepts and we should avoid a simple averaging of predictions from several models. In this chapter, we use a well known and studied ensemble learning algorithm (bagging) and we generate ensembles that make use of several EL techniques proposed in the literature.

4.3 Bagging Workflows

The Ensemble Learning (EL) literature can be split into three main topics: ensemble generation, ensemble pruning and ensemble integration [Mendes-Moreira *et al.*, 2012]. It can also be seen as a process of three phases: 1) generating an accurate and diverse set of models; 2) prune the ensemble in order to decrease its size and attempt to improve its generalization ability; and finally, 3) select a function to aggregate the predictions of each single model of the ensemble. This can be achieved by a static or dynamic method. In the former, one does not take into account the characteristics of the test instance, such as stacking; in the latter, one chooses different subsets of models according to the characteristics of the test instance.

Bagging, one of the most popular EL algorithms, can also be decomposed at the light of the structure that we described above [Breiman, 1996a]. Generically, given a training data set, a sample with replacement (a bootstrap sample) of the training instances is generated. The process is repeated k times and k samples of the training instances are obtained. Then, from each sample, a model is generated by applying a learning algorithm. In terms of aggregating the outputs of the base learners and building the ensemble, typically, bagging uses two of the most common: voting for classification (the most voted label is the final prediction) and averaging for regression (the predictions of all the base learners are averaged to form the ensemble prediction).

In this chapter, we introduce the concept of a bagging workflow, that can also be decomposed into three components: generation, pruning and integration. The following subsections describe some of the methods that can be used within each of these components.

4.3.1 Generation

As mentioned before, bagging generates ensembles by applying a learning algorithm to bootstrap samples of the training data. However, bagging can be regarded as a much more versatile method, if we consider the following choices as instances of hyper-parameters:

- the sampling strategy. Although bootstrap sampling is by far the most common sampling strategy in bagging, there are also reports of interesting results using sub-sampling without replacement and sampling of random subspaces [Ho, 1998].

- the learning algorithm used to generated the models. Decision trees and neural networks are among the favourite, given their unstable learning property [Breiman, 1996a].

- how many models to generate. On the seminal paper in which bagging was introduced [Breiman, 1996a], the author claimed that 50 or 100 single models should be enough to achieve good results. However, more recent studies showed that this problem is highly dataset dependent [Hernández-Lobato *et al.*, 2013].

4.3.2 Pruning

Besides the widespread use among data science practitioners, bagging is also one the most studied algorithms [Bauer and Kohavi, 1999]. One of the discoveries made by researchers is that efficient pruning of a bagging ensemble could lead to a smaller ensemble and also to generalization improvements [Zhou *et al.*, 2002]. This has led a stream of research focused specifically on pruning techniques for bagging ensembles. Since a detailed overview of these techniques is out of scope of this chapter, we refer the reader to some important papers in the field [Martinez-Muñoz *et al.*, 2009; Qian *et al.*, 2015]. Essentially, these techniques combine the concepts of accuracy and diversity in ensemble learning to search for a subset of models that guarantees the same performance of the full ensemble or even improves it. This search procedure is often led by some heuristic or an optimization algorithm.

Therefore, from the ensemble pruning phase of constructing bagging workflows, two hyper-parameters must be considered:

- the pruning method to be used.
- the percentage of models that should be pruned. Again, studies have shown that this is highly dependent on the dataset [Hernández-Lobato *et al.*, 2013].

4.3.3 Integration

As mentioned before, ensemble integration methods are split into two groups: static and dynamic. In the former, the weights assigned to each model in the ensemble are a constant value; in the later, the weights vary according to the instance to be predicted. In the dynamic group, we distinguish between methods for selection (when a single model is selected) or combination of models (when more that one model can be selected).

Regarding static methods, the most well known is stacking [Wolpert, 1992]. Regarding dynamic methods, again, research has shown that this hyper-parameter is highly dataset dependent [Britto *et al.*, 2014]. A large empirical comparison of these techniques can be found in Pinto *et al.* [2016]. A detailed description of these techniques is out of scope for this chapter so we refer the reader to the original papers or to a survey [Cruz *et al.*, 2018].

4.4 autoBagging: Ranking Bagging Workflows

In this section we present *autoBagging*. Although for this work we focused on generating rankings of bagging workflows, we believe that the approach is generic for the algorithm/workflow ranking in ML. Therefore, we describe the method from a generic perspective and provide more specific details on the application to bagging workflows in section 4.5.

We recall Figure 4.1 for a brief overview of the method. We start by describing the learning approach and then how we collected the metadata, both metafeatures and metatarget, to be able to learn at the meta-level.

4.4.1 Learning Approach

We approach the problem of algorithm selection as a learning to rank problem [Liu, 2009; Li, 2011]. Assume $\mathcal{D}$ as the dataset set and $\mathcal{H}$ as the algorithm set. $\mathcal{Y} = \{1, 2, ..., z\}$ is the label set, where each value represents a relevance score, which represents the relative performance of a given algorithm. Therefore, $z \prec z-1 \prec ... \prec 1$, where $\prec$ represents an order relationship.

Furthermore, $\mathcal{D}_q = \{d_1, d_2, ..., d_q\}$ is the set of datasets for training and d_i is the i-th dataset, $\mathcal{H}_i = \{h_{i,1}, h_{i,2}, ..., h_{i,n_i}\}$ is the set of algorithms associated with dataset d_i and $\mathbf{y}_i = \{y_{i,1}, y_{u,2}, ..., y_{i,n_i}\}$ is the set of labels associated with dataset d_i, where n_i represents the sizes of $\mathcal{H}_i$ and $\mathbf{y}_i$; $h_{i,e}$ represents the e-th algorithm in $\mathcal{H}_i$; and $y_{i,e} \in Y$ represents the e-th label in $\mathbf{y}_i$, representing the relevance score of $h_{i,e}$ with respect to d_i. Finally, the meta-dataset is denoted as $S = \{(d_i, \mathcal{H}_i), \mathbf{y}_i\}_{i=1}^{q}$.

We use MtL to generate the metafeature vectors $x'_{i,e} = \phi(d_i, h_{i,e})$ for each dataset-algorithm pair, where $i = 1, 2, ..., q$; $e = 1, 2, ..., n_i$ and ϕ represents the metafeatures extraction function. These metafeatures can describe d_i, $h_{i,e}$ or even the relationship between both. Therefore, taking $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, ..., x_{i,n_i}\}$ we can represent the meta-dataset as $S' = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{q}$.

Our goal is to train a meta ranking model $h'(d, h) = h'(\mathbf{x})$ that is able to assign a relevance score to a given new dataset-algorithm pair d and h, given x.

4.4.2 Metafeatures

We approach the problem of generating metafeatures to characterize d and h with the aid of a framework for systematic metafeatures generation exposed in chapter 3 of this thesis. Essentially, this framework regards a metafeature as a combination of three elements: meta-function, a tuple of arguments and a post-function. The framework establishes how to systematically generate metafeatures from all possible combinations of arguments and post-functions alternatives that are compatible with a given meta-function. Thus, the development of metafeatures for a MtL approach simply consists of selecting a set of meta-functions (e.g. entropy, mutual information and correlation) and the framework systematically generates the set of metafeatures that represent all the information that can be obtained with those meta-functions from the data.

The sets of metafeatures were generated using the following types of arguments:

- `num_feat`
- `cat_feat`
- `cat_target`
- `cat_pred` (from five landmarkers - naive bayes, decision tree with depth 1, 2 and 3, and majority class)
- `dataset`

For this task in particular, we selected a set of meta-functions to characterize the datasets (measuring information regarding the target variable, the categorical and numerical features) the algorithms and the relationship between the datasets and the algorithms (which can be seen as landmarkers [Pfahringer *et al.*, 2000]). Therefore, the set of meta-functions used was:

- skewness - an 1-ary function that takes as input elements of type `num_feat`
- Pearson's correlation - a 2-ary function that takes as input elements of type `num_feat`
- Maximal Information Coefficient (MIC [Reshef *et al.*, 2011]) - a 2-ary function that takes as input elements of type `num_feat`
- entropy - an 1-ary function that takes as input elements of type `cat_feat`, `cat_target` and `cat_pred`
- mutual information - a 2-ary function that takes as input elements of type `cat_feat`, `cat_target` and `cat_pred`
- eta-squared, a measure of effect size for use in ANOVA test - a 2-ary function that takes as input elements of type `cat_target` and `num_feat`
- R value of class overlap [Oh, 2011] - a 1-ary function that takes as input elements of type *dataset*
- average rank method [Brazdil *et al.*, 2009] - a 1-ary function that takes as input elements of type *dataset*

For instance, if we take the example of using entropy as meta-function, it is possible to measure information in categorical features, categorical targets and categorical predictions (if the base-level problem is a classification task). After computing the entropy of all these arguments, it might be necessary to aggregate the information in order to keep the tabular form of the data. Therefore, we choose a palette of aggregation functions to capture several dimensions of these values and minimize the loss of information by aggregation. In that sense, we chose a set of post-functions commonly used in the MtL literature: average, maximum, minimum, standard deviation, variance and histogram binning.

Given these meta-functions, the available arguments and post-functions, we are able to generate a set of 150 metafeatures. To this set, we add three metafeatures: the number of examples of the dataset, the number of attributes and the number of classes of the target variable. Furthermore, we also add four metafeatures to describe each workflow: the number of trees, the pruning method, the pruning cut point and the dynamic selection method [Brazdil *et al.*, 2009]. In total, *autoBagging* uses a set of 157 metafeatures.

4.4.3 Metatarget

To compute the metatarget, we use a cross validation error estimation methodology, in which we estimate the performance of each bagging workflow for each dataset using Cohen's kappa score [Cohen, 1960]. On top of the estimated kappa score, for each dataset, we rank the bagging workflows. This ranking is the final form of the metatarget and it is then used for learning the meta-model.

4.5 Experiments

In this section we describe the experiments performed to understand and evaluate *autoBagging*. We also provide an exploratory analysis of the metadata collected from the experiments that is particularly interesting to understand some of the EL methods used.

4.5.1 Experimental setup

Our experimental setup includes datasets extracted from the OpenML platform for collaborative machine learning [Vanschoren *et al.*, 2014]. We limited the datasets extracted to a maximum of 5000 instances, a minimum of 300 instances and a maximum of 1000 attributes, in order to speed up the experiments and exclude datasets that could be too small for some of the bagging workflows that we wanted to test. However, some datasets were later removed from the experiments due to time constraints. Our final experimental setup comprises 140 datasets. We pre-processed the datasets by deleting the features with more than 15% missing values and/or near zero variance.

Regarding bagging workflows, taking into account all the hyper-parameters described in Section 4.3 would result in a computational cost too large for our resources. Therefore, we limited the hyper-parameters of the bagging workflows to four: number of models generated, pruning method, pruning cut point and dynamic selection method. Specifically, each hyper-parameter can take the following values:

- Number of models: 50, 100 or 200.
- Pruning method: Margin Distance Minimization(MDSQ [Martinez-Muñoz *et al.*, 2009]), Boosting-Based Pruning (BB [Martinez-Muñoz *et al.*, 2009]) or none.
- Pruning cut point: 25%, 50% or 75%.
- Dynamic integration method: Overall Local Accuracy (OLA), a dynamic selection method [Woods *et al.*, 1997]; K-nearest-oracles-eliminate (KNORA-E [Ko *et al.*, 2008]), a dynamic combination method; and none. The k for OLA and KNORA-E was set to 10.

The combination of the hyper-parameters described above generated 63 valid bagging workflows. Decision trees was chosen as learning algorithm since it is the most common algorithm used in bagging ensembles.

We used the *XGBoost* [Chen and Guestrin, 2016] learning to rank implementation for gradient boosting of decision trees to learn the metamodel as described in section 4.4. The decision tree implementation from this library has

a very elegant way of dealing with missing values. Essentially, the tree splitting functionality assigns an instance with missing values to a default direction and then learns from the data the optimal default direction. This is particularly important for MtL since the number of missing values is often quite high in these dataset (e.g., attribute correlation cannot be measured in a dataset without numeric attributes, which results in a missing value).

We compare *autoBagging* with four methods:

- bagging with 100 decision trees
- the average rank (AR) method [Brazdil *et al.*, 2009], which basically is a model that always predicts the bagging workflow with best average rank in the meta training set [2]
- *auto-sklearn* [Feurer *et al.*, 2015], an *autoML* library considered the state-of-the-art in the field
- oracle, an ideal model that always selects the best bagging workflow for each dataset

Note that *auto-sklearn* has a different hypothesis space from the other methods. *autoBagging*, the average rank method and the oracle, all use 63 bagging workflows as hypothesis space; *auto-sklearn* uses several *scikit-learn* methods, including 15 classification algorithms, 14 preprocessing methods and 4 data preprocessing methods. Assuming that all combinations are possible [3], this results into 840 hypotheses. Although we recognize that this makes the task of the search algorithm more difficult, it is also an advantage since the system is not limited to the bias induced by the bagging workflows.

As evaluation methodology, we use an approach similar to the leave-one-out methodology. However, in this case, each test fold consists of a dataset. The remaining datasets were used for training purposes. The evaluation metric used is Cohen's kappa and the methodology proposed by Demšar [2006], with $\alpha = 0.05$, was used for statistical validation of the results. Note that the Cohen's

[2] Recently, an improved version of this method, AR*, has been published [Abdulrahman *et al.*, 2018]. However, for the purpose of this study and benchmarking, we use still we original version, AR.

[3] We did not found information about this in the paper.

kappa is also given to *auto-sklearn* for its internal search procedure. [4] Therefore, all methods are optimising the same evaluation metric.

Finally, an average user of an *autoML* is mostly concerned with predictive performance and execution time. With that in mind, we focus our results analysis on those two perspectives.

4.5.2 Exploratory metadata analysis

Given the rich metadata collected from the experiments that we carried out, we proceed to draw some insights about the datasets and the workflows that we experimented with.

We can see by analysing Figure 4.2 that the range of kappa values for each dataset varies a lot. This is expected given the No Free Lunch theorem, that states that there is no one model that works best for every problem and "two algorithms are equivalent when their performance is averaged across all possible problems" [Wolpert, 1996]. Even though all the models that we experimented with belong to the same family (bagging of decision trees), the pruning and dynamic integration components enable to generate very different predictive models. This is indicative that ranking these bagging workflows for each dataset is not an easy learning task.

Figure 4.3 shows the boxplots of the ranking scores collected for each dataset, ordered by average ranking. We can extract the following insights regarding the bagging workflows:

- on average, the bagging workflows that make use of BB pruning and KNORA-E as dynamic integration method seem to achieve better performance
- in terms of pruning cut point, it seems that BB pruning works better with a large pruning cut point (e.g., 75%) than MDSQ
- the bagging workflows that do not make use of any kind of dynamic integration method are worse on average than the ones that do
- both the top and the worst bagging workflows are outliers for some dataset in terms of performance

[4] All other parameters of the system were set to default.

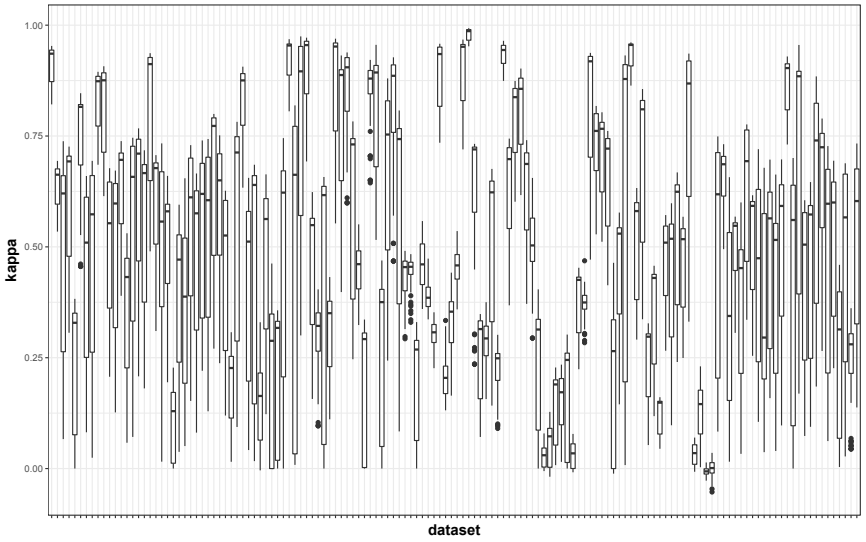

Figure 4.2: Boxlplots of the kappa values collected for each dataset from evaluating the performance of each bagging workflow.

4.5.3 Results

We compared *autoBagging* with four other methods, as mentioned before: bagging with 100 decision trees, the average rank method, *auto-sklearn* and the oracle. We evaluate on validation set three versions of *autoBagging*, taking the top 1, 3 and 5 bagging workflows ranked by the meta-model. For instance, in *autoBagging@3*, we evaluate the top 3 bagging workflows ranked by the meta-model and we choose the best. For the average rank method we only test the top 1 method recommended. The reported results are collected from the test set.

Predictive Performance

Starting by the tail of the CD diagram in Figure 4.4, both the average rank method and *autoBagging@1* show a superior performance than Bagging with 100 decision trees. Furthermore, *autoBagging@1* also shows a superior performance than the average rank method, a difference that is statistically significant. This result can be visualized in Figure 4.6.

Regarding the comparison with *auto-sklearn*, we can see that *autoBagging@1* has a slightly worst performance. However, that difference is not statistically

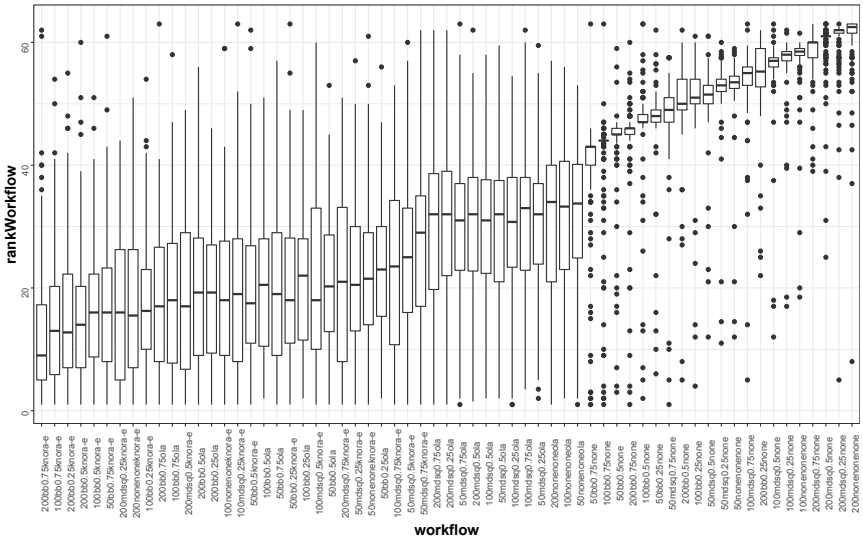

Figure 4.3: Boxplots of the ranking scores collected for each bagging workflow. For instance, *200bb0.75knora-e* represents a bagging workflow with 200 trees, to which boosting-based pruning is applied with a 75% cut point and KNORA-E is used as dynamic integration technique.

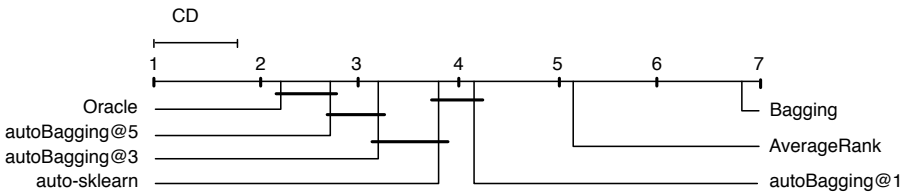

Figure 4.4: Critical Difference diagram (with $\alpha = 0.05$) of the experiments.

significant. If we compare *autoBagging@3* with *auto-sklearn* we can see a better performance by the former. Again, the difference is not statistically significant.

The CD diagram in Figure 4.4 also shows that *autoBagging@3* and *autoBagging@5* have a similar performance. However, and we must highlight these results, *autoBagging@5* shows a performance that is not statistically different from the oracle. This is extremely promising since it shows that the performance of *autoBagging* excels if the user is able to test the top 5 bagging workflows ranked by the system.

In the comparison between *autoBagging* and *auto-sklearn*, it is important

to stress that the results are somewhat surprising, given that the bias of our method is much stronger than the bias of *auto-sklearn*. We hypothesize that one of the reasons for this result is the fact that *auto-sklearn* executes an internal split of the training set provided to acquire a validation set. This set is then used to guide the search procedure. In small datasets (and all datasets in our experimental setup have less that 5000 examples), this can easily lead to overfitting. We plan to further study this issue in future work.

Execution Time

As mentioned before, the average user of an *autoML* system is interested not only in the predictive performance of the system but also in its execution time. [5] With this in mind, we collected the execution time of all methods included in our experimental setup. The only method that allows to set a time limit is *auto-sklearn*, the default value being one hour. Therefore, we compare the execution time of *autoBagging* and the average rank method in regard to this setup.

Figure 4.5 shows a visualization of the distribution of the execution time for each method in the 140 datasets. We can see that fastest method is clearly the average rank, as it was expected. However, the average difference in comparison with *autoBagging@1* is small and it is due to the overhead of computing the metafeatures required by *autoBagging*.

Naturally, the execution of *autoBagging@3* and *autoBagging@5* increases as the number of bagging workflows to train also increases. Again, the differences in execution time are small.

Finally, regarding the comparison with *auto-sklearn*, we can see that only two runs (one for *autoBagging@3* and another for *autoBagging@5* - and they both belong to the same dataset) surpass the time limit of 3600 seconds as used in the *auto-sklearn* experiments. Take for instance *autoBagging@1*, our method presents a median execution time lower by a factor of almost 6 than the one presented by *auto-sklearn*.

[5] By execution time we mean the time in seconds that system takes to select the top@k algorithm(s) and train the model(s).

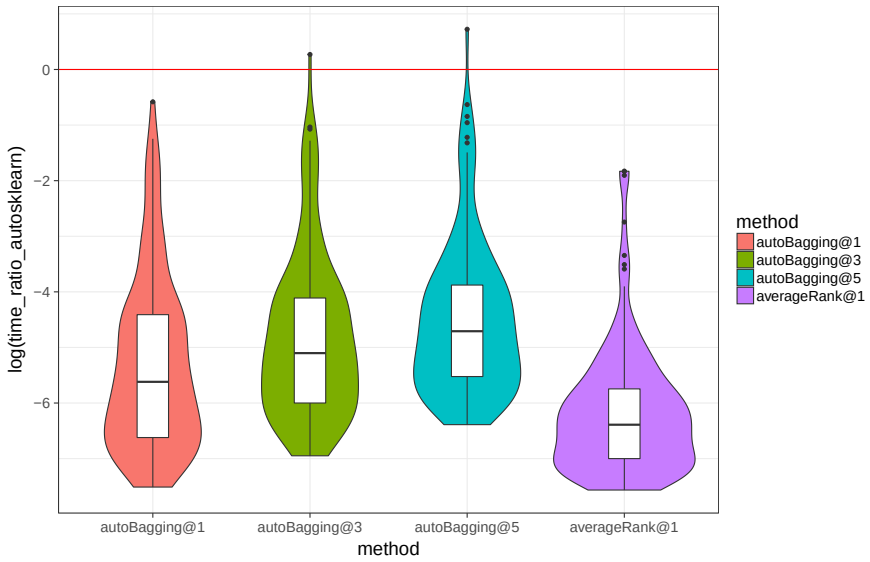

Figure 4.5: Violin and boxplots showing the distribution of execution time for the method *autoBagging@1, autoBagging@3*, *autoBagging@5* and *averageRank@1* in comparison with *auto-sklearn*. We computed the ratio of the execution time of each method in seconds by the execution time of auto-sklearn (3600 seconds). The logarithmic transformation was applied for visualization purposes. The red line represents the execution time of *auto-sklearn*, since $\log(1) = 0$.

Comparison of Ranking Methods

A loss curve shows the relationship between the average loss in terms of performance with the number of workflows tested, according to the rank suggested by each method. The loss is calculated as the difference between the performance of the best top@K algorithm in comparison with the ground truth ranking. The loss for all datasets is then averaged for aggregation purposes. Figure 4.6 is an example of such loss curve. We can see, as expected, that the average loss decreases for both methods as the number of workflows tested increases.

Since *autoBagging* and the average rank method are the only ranking methods in the experimental setup, we only show the loss curve for these methods. Therefore, in Figure 4.6, it is possible to visualize that *autoBagging* shows better performance for all the values of the x axis. Interestingly, this result is particularly noticeable in the first tests. For instance, if we test only the top 1 workflow

recommended by *autoBagging*, on average, the kappa loss is half of the one we should expect from the suggestion made by the average rank method. However, we must state that *autoBagging* still has room for improvement. As one can see in Figure 4.6, it takes our method 20 workflows to achieve a performance within 10% of the best performance and 50 workflows to achieve the best.

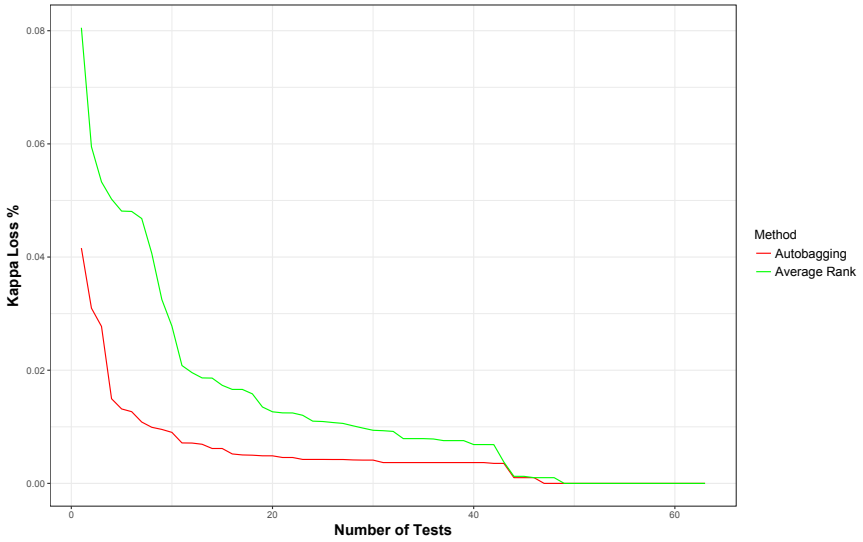

Figure 4.6: Loss curve comparing autoBagging with the Average Rank method.

In Figure 4.7 we statistically validate the results observed in the loss curve. Using MAP@10 as evaluation metric at the meta-level, *autoBagging* presents a clearly superior performance in comparison with the average rank method.

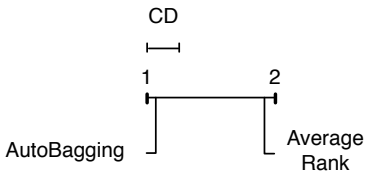

Figure 4.7: Critical difference diagram comparing autoBagging with the Average Rank method.

Metafeatures Importance

Figure 4.8 shows the relative importance of the top 30 most important metafeatures. It is clear that the most informative metafeatures are the ones generated using the rank of each workflow in the meta-training set as meta-function. This result indicates that *autoBagging* is complementing the useful information in the average rank method with more specific knowledge that allows it to generalize to cases that deviate from the average. Therefore, the remaining metafeatures are critical for the ability of the meta-model to generalize for all datasets.

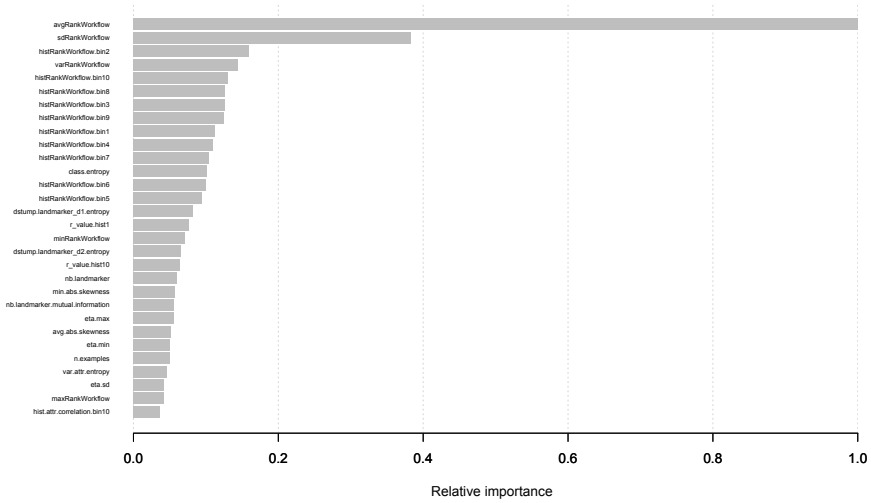

Figure 4.8: Top 30 most important metafeatures for the *XGboost* metamodel measured using *Gain*, which represents the relative contribution of the corresponding feature to the model calculated by taking each feature's contribution for each tree in the model.

Metafeatures such as *class.entropy*, *dstump.landmarker_d1.entropy* and *r_value.hist1*, are also among the most informative metafeatures. Metafeatures such as these ones are critical for the ability of the meta-model to generalize for all datasets.

4.6 Conclusions and Future Work

This chapter presents *autoBagging*, an *autoML* system that makes use of a learning to rank approach and MtL to automatically suggest a bagging ensemble specifically designed for a given dataset.

We tested *autoBagging* on 140 classification datasets and the results show that our method has a very competitive performance in comparison with other state-of-the-art methods, such as *auto-sklearn*. In fact, if the top five workflows suggested by *autoBagging* are tested, results show that the system achieves a performance that is not statistically different from the *oracle*, a method that systematically selects the best workflow for each dataset. For the purpose of reproducibility and generalizability, *autoBagging* is publicly available as an R package.

As future work, we plan to further improve the experimental setup of *autoBagging* by comparing it with other approaches such as the hyperband algorithm. Furthermore, we plan to study how we can use bayesian optimization to further improve the final ensemble, always taking into account concepts such as diversity and complementarity between models to design the final ensemble.

Part III

Metalearning for Ensemble Learning

Chapter 5

Analysing and Pruning Bagging Ensembles with Metalearning

5.1 Introduction

Ensemble learning (EL) refers to methods that combine several models to make a final prediction, typically in a classification or regression scenario. The EL literature can be split into three main topics: ensemble generation, ensemble pruning and ensemble integration. In this work, we make contributions to two of these topics, ensemble generation and ensemble pruning, and we will focus specifically on the bagging algorithm. Our approach for both topics is based on Metalearning (MtL) techniques. In the former, we use MtL from a descriptive perspective to gain a better understanding the performance and intrinsic behaviour of the bagging algorithm; in the latter, MtL is used in a predictive approach, in which we generate a metamodel that is able to predict if a bootstrap sample has *good* characteristics and, thus, decide if it should be pruned or not from the final ensemble.

As one of the most used EL algorithms, understanding bagging, both from a theoretical and empirical perspective, is very important for its successful applica-

tion by ML practitioners. Furthermore, in part due its simplicity and scalability, bagging has led to the proposal of several methods that aim to improve the algorithm somehow, particularly for pruning bagging ensembles [Martinez-Muñoz *et al.*, 2009].

We propose and apply an empirical methodology to study bagging performance. We investigate the reasons that affect the influence of a bootstrap (and corresponding model) in the space of sub-ensembles. For that, we compute specific bootstrap characteristics. These measures are then compared with the importance of a bootstrap [1] on the predictive performance of ensembles that include the model generated by applying a learning algorithm to it.

We tested our proposed methodology by executing experiments with 53 classification datasets collected from the UCI repository [Blake and Merz, 1998]. For each dataset, we generated bagging ensembles of decision trees with 20 and 100 models. We were able to generate and test all possible combinations of the ensembles with 20 models. However, for computational reasons, we were forced to sample the number of combinations tested for ensembles with 100 models. We present results that indicate the validity of this sampling procedure. All the insights collected from the metadata describing ensembles with 100 models are compared with the 20 models case. This allowed a validation of our sampling procedure. Given the descriptive aim of our work, we used standard exploratory data analysis procedures to extract knowledge from the metadata that we generated.

Furthermore, we propose a MtL method that makes use of the metadata collected from the experiments described above to generate a metamodel that is able to prune bagging ensembles based on the characteristics of the bootstrap samples. Our approach differs from the other ensemble pruning methods in the sense that allows to prune the ensemble by just analysing the characteristics of a bootstrap sample (or metafeatures, in MtL nomenclature) and before actually generating the individual models. For that, we adapted several metafeatures already proposed in the literature and we also introduce some new ones that are very specific of our problem domain [Brazdil *et al.*, 2003; Pfahringer *et al.*,

[1] We define an important bootstrap as a bootstrap which its correspondent model belongs to the best combinations of tested ensembles in terms of performance.

2000].

The contributions of this chapter are: 1) a methodology based on an extensive experimental procedure and on MtL for empirically studying the performance of bagging; 2) new metafeatures that characterize the relationship between bootstrap samples and the complete training data; 3) an exploratory MtL approach using visualization and a statistical method applied to 53 UCI classification datasets, yielding interesting observations concerning the relationship between the characteristics of the bootstrap sample and the performance of the bagging ensemble; 4) a MtL method for pruning bagging ensembles 5) comparison of different methods for ensemble pruning in 53 UCI classification datasets.

This chapter is organized as follows. Section 5.2 describes the related work with the main contributions of this work, specifically regarding the understanding of the bagging algorithm and pruning methods for bagging ensembles. Section 5.3 presents the empirical methodology for studying ensembles and a study of the representativeness of the results obtained by sampling from all the possible ensembles with 100 models. Section 5.4 describes the MtL approach used in this work as well as the metafeatures. In section 5.5, we present the descriptive study on the characteristics of a bootstrap and its importance on the predictive performance of an ensemble. Section 5.6 provides details about the MtL pruning method and presents the results regarding those experiments. Finally, section 5.7 concludes the chapter with some final remarks and future work.

5.2 Related Work

In this section we present the related work both regarding the papers that aim a better understanding of bagging or propose pruning techniques for bagging ensembles. Since, to the best of our knowledge, we are the first to use MtL for those purposes, we focused mostly on related work in terms of approaches and techniques that aim the same goals but do not use MtL.

5.2.1 Understanding bagging

Several papers propose theoretical frameworks that provide important insights on the effectiveness and reasons behind the success of bagging. Breiman [1996a] argued that aggregating can transform good predictors into nearly optimal ones, highlighting however the importance of using unstable learners (small variations in the training set must generate very distinct models [Breiman, 1996b]).

Friedman [1997] related bagging with the bias and variance decomposition of the error. Shortly, the error is split into two components: bias, associated with the intrinsic error of the learner generalization ability; and variance, associated with the error assigned to the variation in the model from one bootstrap to another. In the context of bagging, Friedman claimed that the variance component is reduced (because of the bootstrapping procedure) without changing the bias.

Domingos [1997] presented two alternative hypotheses for the success of bagging: although rejecting the possibility of approximation to the optimal procedure of Bayesian model averaging with an appropriate implicit prior probability distribution, he proved that bagging works effectively because it shifts the prior to a more appropriate region of model space. However, Domingos [1997] recognized one important fact: none of the above frameworks relate the success of bagging with the domain characteristics.

Friedman and Hall [2007] confirmed Breiman [1996b] claim by showing that bagging is most successful when used with highly non-linear estimators such as decision trees and neural networks. In this study they also found evidence that sub-sampling is virtually equivalent to traditional bootstrap sampling. Büchlmann and Yu [2002] provided theoretical explanations of the same claim.

Grandvalet [2004] provided an interesting study in which he found that bagging equalizes the influence of examples in a predictor. Bootstrapping a dataset implies that fewer examples have a small influence, while the highly influential ones are down-weighted. The author claims that bagging is useless when all examples have the same influence on the original estimate, is harmful when high impact examples improve accuracy, and is otherwise beneficial.

5.2.2 Pruning bagging ensembles

For classification, ensemble pruning methods are often inspired by the ensemble learning diversity literature. That is, the methods focus on searching for complementary classifiers. Margineantu and Dietterich [1997] showed firstly that there is no need for all the classifiers in a boosting ensemble. The development of ensemble pruning methods are often biased towards bagging since it is noted that these kind of methods are more effective with bagging than with boosting [Hernández-Lobato *et al.*, 2006].

In terms of the nature of the methods, we can see two different research directions: one focused on optimization based methods and another on ordering based methods. In the former, the methods use an optimization technique to select a subset of models. Zhang *et al.* [2006] approached ensemble pruning as a quadratic integer programming problem that is solved by applying semi-definite programming to a convex relaxation of the original problem. Qian *et al.* [2015] proposed a method inspired by a multi-objective evolutionary algorithm. In the latter, the methods start with an empty set and gradually add models in order to minimize/maximize a certain objective. This iterative process allows to generate an order of the individual models. Martinez-Muñoz *et al.* [2009] published a detailed analysis of these kind of methods for bagging ensembles. We use one of the methods (the one with the best overall results in Martinez-Muñoz *et al.* [2009] - *MDSQ*) proposed by them for comparison with our method. Li *et al.* [2012] present a theoretical study of diversity for ensembles of classifiers and proposed a greedy forward pruning method that exploits their discoveries.

Another important question regarding ensemble pruning that is relevant for most of the methods is the size of the pruned ensemble. Hernández-Lobato *et al.* [2013] empirically showed that the optimal ensemble size is very sensitive to the particular classification problem considered. However, for a wide range of classification problems, a pruning of 60-80% seems appropriate [Martinez-Muñoz *et al.*, 2009].

5.3 Empirical Methodology to Characterize Bagging Performance

Formally, an ensemble EH gathers a set of predictors of a function h denoted as $\hat{h}_i$. Therefore, $EH = \{\hat{h}_{i=1,\ldots,k}\}$ where the ensemble predictor is defined as $\hat{EH}_h$.

We propose a methodology to empirically analyse the behaviour of bagging. Given a set of k bootstrap samples (also referred to in this chapter as bootstraps, for simplicity), we estimate the empirical distribution of performance of the bagging ensembles that can be generated from all elements of its power set. In other words, we estimate the empirical distribution of performance of all possible ensembles of size 2, 3, ... k that can be generated from those k bootstraps.

This distribution can be used to study the role of a given bootstrap (and respective predictive model $\hat{h}_i$) in the performance of $2^k - 1$ possible ensembles, as done in this chapter. Additionally, the distribution can be used to analyse the joint relationship between the bootstrap samples in each ensemble and its performance.

It is easy to understand that is impossible to execute the complete set of experiments for ensembles with a realistically large size, such as k=100, given that the number of combinations to test is $2^k - 1$. Therefore, the only possibility is to estimate the distribution of the performance of all ensembles that can be generated with the set of k bootstraps by sampling from its power set. To investigate the validity of this approach, we carried out the following study.

5.3.1 Estimating the distribution of performance by sampling

To validate our methodology based on sampling, we executed the full methodology with k=20 and then we studied the impact of sampling. Based on these results, we extrapolate our findings for k=100. We used the Kullback-Leibler Divergence (KLD) ([Kullback and Leibler, 1951]) to measure the difference between the probability distributions Q and S, defined as

$$\Phi_{KL}(Q||S) = \sum_i Q_i log_2 \left(\frac{Q_i}{S_i}\right) \tag{5.1}$$

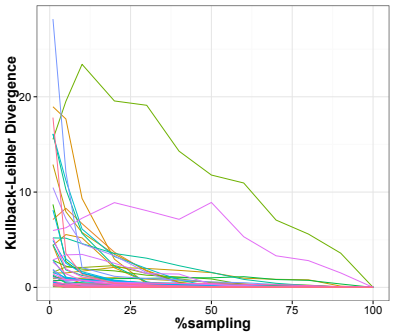

Figure 5.1: KLD between % of sample and population. Each line represents a different dataset.

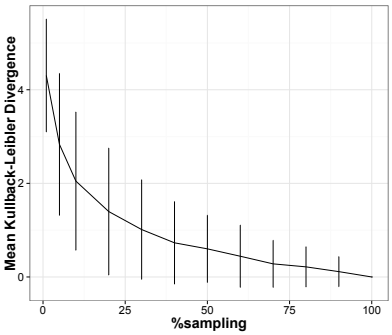

Figure 5.2: Mean KLD (and standard deviation) between % of sample and population.

where Q is the results obtained by testing $2^k - 1$ combinations of k models and S a sample of those results. Since the KLD measure is not symmetric, we averaged the divergences, then

$$\bar{\Phi}_{KL} = \frac{\Phi_{KL}(Q||S) + \Phi_{KL}(S||Q)}{2} \tag{5.2}$$

Given that this experiment implies a large component of randomness, we executed each sampling procedure 100 times and we averaged the values obtained.

In the first experiment, for each dataset, we progressively increased the sampling proportion and systematically computed the KLD between the sample and the population with k=20. Figure 5.1 shows, as expected, that as the sampling proportion increases, the divergence between the samples and respective population decreases. One can see that for most of the datasets the fall of the curve is rather fast. Figure 5.2 shows the same result but the values for the 53 datasets are averaged for each sampling proportion. Again, as expected, the standard deviation and the mean KLD decreases as the proportion of sampling increases.

To assess the hypothesis that increasing the number of models in an ensemble changes the sampling results, we repeated the experiment for ensembles with different k values, from 10 to 19. Figure 5.3 shows a slight increase in the divergence between the samples of equal proportion and respective populations as k increases. This result is expected given that the introduction of a new model can possibly change the inter-relations between the models and therefore

affect the performance of some subsets of models. However, all the curves[2] present a very similar pattern. This is indicative that a similar curve could be assumed for an ensemble with k=100.

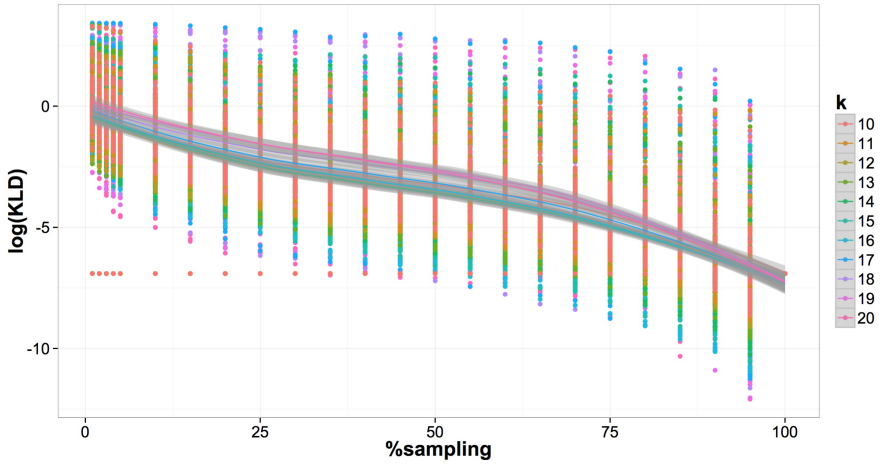

Figure 5.3: Sampling and Kullback-Leibler Divergence, averaged for all datasets.

5.3.2 Discussion

Although the evidence showed previously gives us confidence in the sampling variant of our methodology, we still lack sensitivity on the KLD measure to be able to interpret the values of this experiment more reliably. It is difficult by just looking to the graphs if we are actually losing significant information by sampling.

Figures 5.4 and 5.5 show two density graphs for a 10% sample and the corresponding complete population. The first concerns the *dis* dataset. One can see that even for a very large divergence (30.89), the distribution of the sample is very similar to the population distribution. The second graph concerns the *acetylation* dataset, which has a lower divergence (0.23). Most of the datasets show similar values of divergence between their samples and respective populations. This is indicative that we can sample the performance of an ensemble with k=100 and proceed our study.

The shape of the density graphs is also an interesting result. Both graphs

[2]Estimated using a Local Polynomial Regression (LOESS).

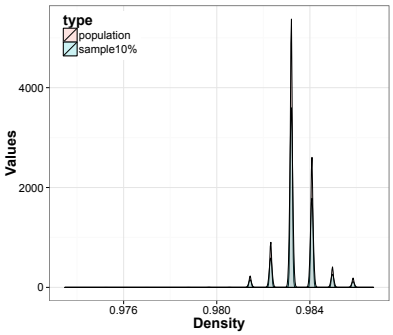

Figure 5.4: Density plot for a 10 % sample and population of the *dis* dataset. The KLD between this sample and population is 30.89.

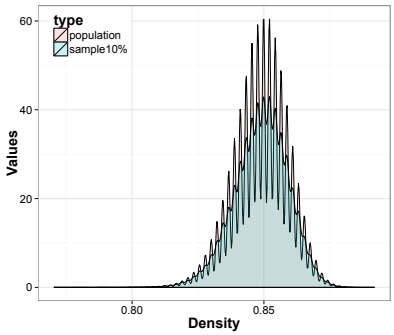

Figure 5.5: Density plot for a 10 % sample and population of the *acetylation* dataset. The KLD between this sample and population is 0.23.

presented a very peculiar pattern of multiple peaks. This is explained by the fact that the bagging performance is a discrete variable. The number of accuracy values that is possible to achieve with all the combinations of a finite set models is limited.

5.4 Generating Metadata

The methodology presented in section 5.3 can be used to provide insights on the types of bootstraps, in terms of how they contribute to the performance of the ensemble. Additionally, it can be combined with a MtL approach to analyze the relationship between the characteristics of the bootstrap sample and the performance of the ensemble.

For that purpose, since the goal in centred on the interpretability of the metafeatures, we relied on simple, statistical, information-theoretic and landmarker metafeatures. For the first group, we selected several metafeatures already present in the literature which were first used for MtL in the METAL and Statlog projects [Brazdil *et al.*, 2009]. We also introduce a new metafeature based on the Jensen-Shannon distance between a bootstrap and the training set [Lin, 1991] . This metafeature aims to measure how different is the bootstrap from the original dataset. This metric has proved to be useful while measuring stability in multi-source data [Saez *et al.*, 2013]. It can also be seen

as a diversity measure that focuses directly on the bootstrap sample and not on the predictions made by the generated model.

We used two landmarkers: a decision stump and a Naive Bayes classifier. Given the different bias of the algorithms, it is expected that the metafeatures can help capture different patterns. We also used two diversity measures proposed in the ensemble learning literature: the Q-Statistic [Kuncheva and Whitaker, 2003] and Classifier Output Difference [Peterson and Martinez, 2005] (COD) measures. Kuncheva and Whitaker [2003] state that the Q-Statistic is the diversity measure with greater potential for providing useful information about ensemble performance.

We adapted the Q-Statistic to the specificities of our problem. Kuncheva and Whitaker [2003] present it as a metric to measure the diversity of an ensemble. We use it to measure the diversity between the predictions of two models: one generated by applying a learning algorithm to a bootstrap (b) and the other to the original dataset (d). Using such a measure in this study gives a different perspective on its usefulness. Formally, our adapted Q-Statistic is

$$QS_{b,d} = \frac{C^{cc}C^{ww} - C^{wc}C^{cw}}{C^{cc}C^{ww} + C^{wc}C^{cw}} \tag{5.3}$$

where each element is formed as in Table 5.1.

Table 5.1: Relationship between a pair of classifiers in a bootstrap b and a dataset d.

	f_b**correct**	f_b**wrong**
f_d**correct**	C^{cc}	C^{cw}
f_d**wrong**	C^{wc}	C^{ww}

Lee and Giraud-Carrier [2011] published a paper on unsupervised MtL in which they study the application of several diversity measures for ensemble learning as a distance function for clustering learning algorithms. In their experiments, only one measure, COD, presents results that indicate that it can be a good measure for estimating the potential of combining classifiers. We found this indicative that the metric can also be useful in our problem.

In summary, the metafeatures used for this analysis are: *number of examples* of a bootstrap, *number of attributes*, *proportion of symbolic attributes*, *proportion of missing values*, *proportion of numeric attributes with outliers*, *class en-*

tropy, *average entropy between symbolic attributes*, *average mutual information between symbolic attributes and the class*, *average mutual information between pairs of symbolic attributes*, *average absolute correlation between numeric attributes*, *average absolute skewness between numeric attributes*, *average kurtosis between numeric attributes*, *canonical correlation* of the most discriminating single linear combination of numeric attributes and the class distribution, *Jensen-Shannon distance* between the dataset and bootstrap, decision stump *landmarker*, Naive Bayes *landmarker*, Q-Statistic and COD.

The experiments that we carried with the UCI datasets allowed to collect results from the performance of the bagging algorithm in very distinct learning problems. Given that our goal is to understand the importance of each model (and respective bootstrap) in the ensemble space, we need to aggregate the results obtained for each one of them and compute an estimate of importance.

We use the measure NDCG [Järvelin and Kekäläinen, 2002] to form our metatarget. We consider the performance values of each bootstrap b in all combinations of models in which the bootstrap participated. Then, by applying the NDCG formula we are able to generate an estimate of importance for each bootstrap.

5.5 What Makes a Good Bootstrap?

In order to enable a more concise exploratory analysis of the metadata, we discretized the metatarget. This process is done using the Fisher-Jenks algorithm [Fisher, 1958]. The method was chosen since it is well suited to find the optimal partition into different classes of a continuous variable.

Most of the metafeatures described characterize the bootstrap in isolation. For instance, the *class entropy* metafeature focuses on the bootstrap and does not relate it with the original dataset. One exception is the diversity measure that characterizes the difference between a set of predictions from a model learned on a bootstrap and another model learned in the original training set. Furthermore, some metafeatures computed for bootstraps of the same dataset show very similar values. For instance, it is not expected that the class entropy varies significantly across bootstrap samples of the same training set. Addition-

ally, the range of values of a metafeature for different datasets is expected to be quite different. However, we need metafeatures with values in comparable ranges across datasets to be able to extract useful insights with our MtL approach. In summary, we need to transform the metafeatures in order for them to 1) discriminate between bootstrap samples from the same dataset and 2) be comparable across datasets.

So, we applied one of two simple transformations to each meta-variable: **1**) proportional difference of the metafeature computed for the bootstrap in relation to the metafeature computed for the original training set **2**) proportional difference of the metafeature computed for the bootstrap in relation to the maximum value computed for all the bootstraps of the dataset. Then, it is rescaled in order to keep the natural interpretation of the variables by subtracting 1.

The first transformation was applied to all the metafeatures except the Jensen-Shannon distance, Q-Statistic and COD. To these metafeatures, since we could not compute them in original training set, we applied the second transformation.

The results of the discretization of the metatarget can be verified in Figures 5.6 and 5.7. One can see that the discretized values are grouped in a descending order of the value of the metatarget, as it is desirable. Through the analysis of the results we will mention the concept of importance. We consider that bootstraps of class A are more important than bootstraps of class B or C, therefore, we are interested in understanding the characteristics of important bootstraps.

However, some classes group very few observations. It can become problematic to analyse those groups. We decided to merge these classes and reduce the sparsity of the discretization. The graphs at the bottom of Figures 5.6 and 5.7 show the boxplots of the metatarget variable after that rearrangement.

5.5.1 Exploratory analysis

To assist our analysis, we used Kruskal-Wallis one-way analysis of variance with Wilcoxon pairwise rank sum test as post hoc procedure (0.95 confidence interval) with Holm adjustment method. This analysis was carried to check for significant different medians of the metafeatures among the classes of the metatarget. Fig-

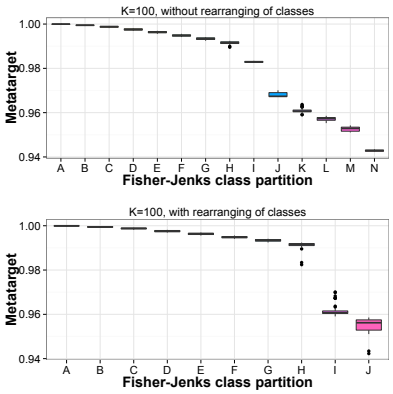

Figure 5.6: Boxplot of numeric metatarget (k=100) vs classes found by Fisher-Jenks algorithm.

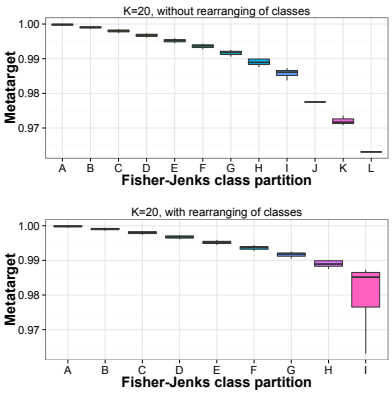

Figure 5.7: Boxplot of numeric metatarget (k=20) vs classes found by Fisher-Jenks algorithm.

ures 5.8 and 5.9 show the results of Wilcoxon test for the metafeatures that the Kruskal-Wallis test showed a *p-value*below 0.05. One can see that the metafeatures *avg.symb.pair.mutual.information*, *nb.landmarker* and *q.statistic* are the most discriminative ones. We will focus on metafeatures that are more interesting for the ensemble learning literature and withdraw the analysis of the remaining metafeatures due to space limitations.

The Jensen-Shannon distance shows a very interesting pattern that can be verified in Figure 5.10. One can see that the gradient of the colours associated with each class (in descending order of importance) is reflected in the density distribution graphs. If we compare the distribution of the most important bootstraps (classes A, B, C...) with the less important ones it is clear that, as the Jensen-Shannon distance decreases, the importance of the bootstraps associated with that value also decreases. In other words, bootstraps that are very similar with the original training dataset do not generate a useful model for a bagging ensemble. This is not new for the ensemble learning literature, however, here we measure diversity without any learning process involved. However, this result can not be verified in Figure 5.11 which represents the metadata with k=20. This can be explained by the fact that since the k=20 experiment generates fewer bootstraps it is harder to find bootstraps with low importance (we can see in Figure 5.7 that the range of the metatarget in this experiment is smaller

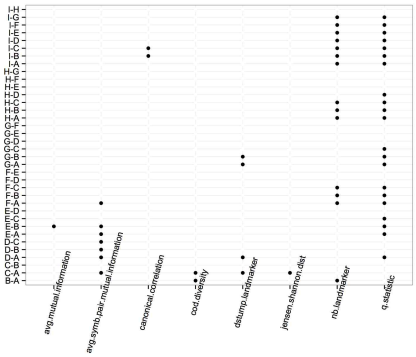

Figure 5.8: Pairwise Wilcoxon Rank Sum test for multiple comparison procedures (k=20). Black dot represents a significative difference between the pair of classes.

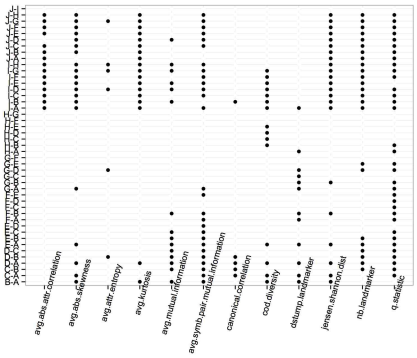

Figure 5.9: Pairwise Wilcoxon Rank Sum test for multiple comparison procedures (k=100). Black dot represents a significative difference between the pair of classes.

than in the k=100 experiment). However, this remains to be confirmed, which could be done by repeating these experiments for other values of k.

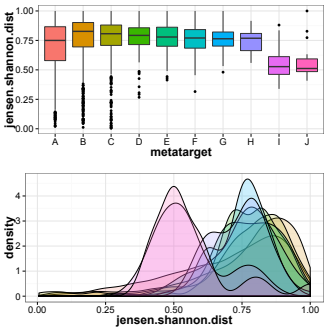

Figure 5.10: Boxplot and density distribution of the Jensen-Shannon distance with k=100.

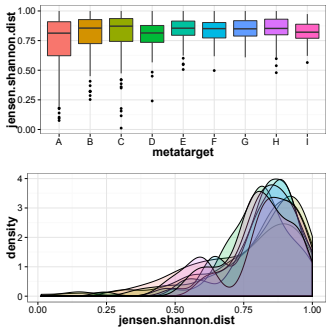

Figure 5.11: Boxplot and density distribution of the Jensen-Shannon distance with k=20.

Figures 5.12 and 5.13 show the density distribution of the diversity measures along the classes of the metatarget. Concerning the Q-Statistic (the bigger, the lesser is the diversity), the results are unclear. Although the Wilcoxon test shows that this metafeature has discriminating power, that is not visible graphically. The values of all classes are extremely biased to 1. This may seem contradictory

to existing knowledge in the ensemble learning literature, where the Q-Statistic is known to be a good diversity indicator [Kuncheva and Whitaker, 2003].

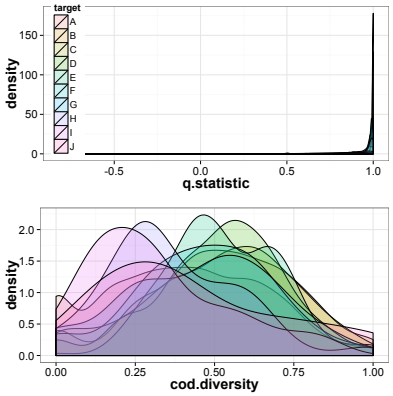

Figure 5.12: Density distribution of the metafeatures Q-Statistic and COD for the k=100 experiment.

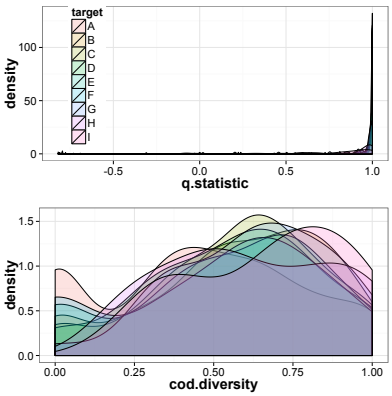

Figure 5.13: Density distribution of the metafeatures Q-Statistic and COD for the k=20 experiment.

However, we must note that the Q-Statistic is usually computed between models of bootstrap samples while in our case, it is between models of a bootstrap sample and the training set. On the other hand, the COD metric (higher the value, higher is the diversity) shows a very clear direct relationship between diversity and importance of a bootstrap in the k=100 experiment. Again, the result is not confirmed by the k=20 graph. However, we consider this result indicative of the effectiveness of this measure in estimating the potential of combining two classifiers.

Finally, by analysing the landmarker metafeatures in Figures 5.14 and 5.15 we can see some interesting patterns. The most prominent one is that important bootstraps have a very similar predictive performance using naive algorithms (such as Naive Bayes and Decision Stump) by comparison against the training set: since we transformed this metafeature as explained previously, a negative value means that the bootstrap has a greater predictive performance than the training set and a positive value the exact opposite. Moreover, we can also see a protuberant peak of the classes that gather the worst bootstraps in the density curves at the left side of the graphs. This indicates that *bad* bootstraps have a superior predictive performance than the training sets.

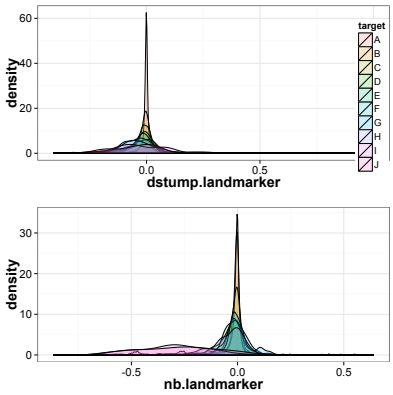

Figure 5.14: Density distribution of the landmakers Decision Stump and Naive Bayes for the k=100 experiment.

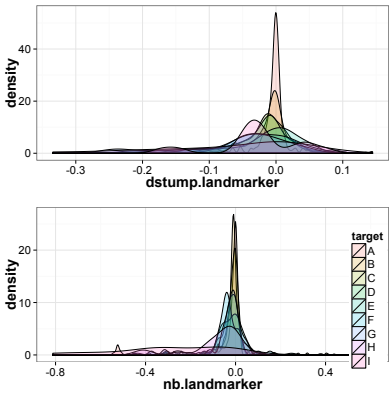

Figure 5.15: Density distribution of the landmakers Decision Stump and Naive Bayes for the k=20 experiment.

5.6 Pruning Bagging Ensembles with Metalearning

The related work in section 5.2 regarding pruning techniques for bagging clarifies that, to the best of our knowledge, all methods require that the individuals models of the ensemble have been generated before applying the pruning technique. Most of these requires are heuristics or optimization procedures that search for a subset of models that combined achieve a good performance both in terms of accuracy and diversity.

Our proposal is a MtL method for pruning bagging ensembles that does not require *a priori* the generation of models: the models are *pre-pruned* just by analysing the characteristics of the respective bootstrap samples. The approach is summarized in Figure 5.16. We generate a metamodel that takes as input the metafeatures computed from the bootstrap samples and outputs a score of relevancy of the bootstrap sample. The metadata is generated using the methodology described previously in Section 5.4.

The sets of metafeatures were generated using the following types of elements:

- `num_feat`

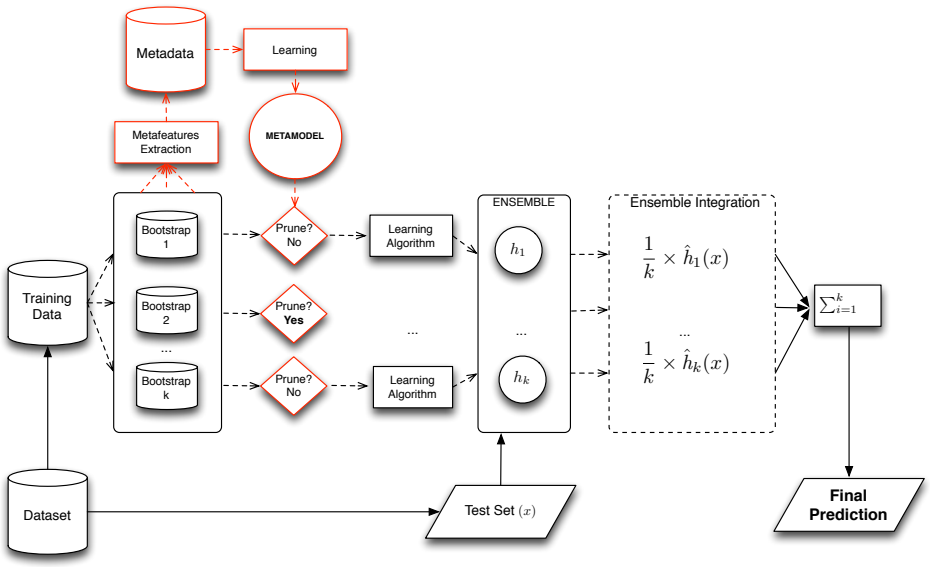

Figure 5.16: Schema of the approach for pruning bagging ensembles with MtL.

- `cat_feat`
- `cat_target`
- `cat_pred` (from five landmarkers - naive bayes, decision tree with depth 1, 2 and 3, and majority class)
- `dataset`

For the purpose of this task, we systematized the generation of the set of metafeatures using the approach described in chapter 3 using the following meta-functions:

- entropy - an 1-ary function that takes as input elements of type `cat_ feat`, `cat_target` and `cat_pred`
- mutual information - a 2-ary function that takes as input elements of type `cat_ feat`, `cat_target` and `cat_pred`
- Pearson's correlation - a 2-ary function that takes as input elements of type `num_feat`
- skewness - an 1-ary function that takes as input elements of type `num_feat`

- kurtosis - an 1-ary function that takes as input elements of type `num_feat`
- Jensen-Shannon distance - a 1-ary function that takes as input elements of type *dataset*

As post-processing functions, we used the following set: average, maximum, minimum, standard deviation, variance and histogram bins.

The resulting set of metafeatures extracted with the framework was joined with the set described in section 5.4 (excluding repeated metafeatures). As metatarget for learning the meta-model, we used the one as described in section 5.4, without any post-processing.

5.6.1 Experimental setup

The system was evaluated with leave-one-out cross-validation. However, in each fold, instead of leaving a single instance for testing, all the instances associated with the same dataset are used for testing. This procedure allows to train the meta-model in meta-data from 52 datasets and test the approach in another dataset. The final error estimation is computed by averaging the results on the 53 datasets.

At the meta-level, since the target is a numeric variable, the evaluation metric chosen is the *Root Mean Squared Error*; at the base-level, the evaluation metric chosen is accuracy.

All the experiments were carried in the *R* software [R Core Team, 2012], using the package *party* for generating the decision trees. We used RReliefF [Robnik-Šikonja and Kononenko, 2003] to assist the selection of metafeatures both for k=20 and k=100.

Meta-learners

We used three learning algorithms to generate the meta-models: M5' Wang and Witten [1997] (*Meta.M5'*), Support Vector Machines with radial basis kernel function Cortes and Vapnik [1995] (*Meta.SVM*) and Random Forests Breiman [2001a] (*Meta.RF*). The performance of the meta-learners is going to be compared with a baseline: the average of the metatarget in the (meta) training data.

Benchmark Pruning Methods

We compare our method with 4 alternatives:

- *Metatarget.* In this approach we use the groundtruth of our metatarget to execute the pruning at the basel-level. This allows to benchmark how good our method could be if we were able to generate an idealistic meta-model.
- *Bagging.* The same algorithm proposed by Breiman [1996a], without any sort of pruning.
- *Margin Distance Minimization* (*MDSQ*, Martinez-Muñoz *et al.* [2009]). This algorithm belongs to the family of pruning methods based on modifying the order in which classifiers are aggregated in a bagging ensemble. The main feature of these kind of methods is to exploit the complementariness of the individual classifiers and find a subset with good performance. Results presented by the authors show that *MDSQ* can greatly reduce the size of the ensemble without significant loss of generalization ability (for some datasets an improvement of the results over bagging was verified).
- *Random pruning.* A baseline approach in which the selection of models to be pruned is random. This is repeated 30 times for robust results.

Ensemble Size

As mentioned previously, determining the optimal ensemble size is not a trivial task. [Hernández-Lobato *et al.*, 2013] show that the optimal size of a bagging ensemble for a dataset is very specific to the particular classification problem considered. However, taking into account their results in 25 datasets, we concluded that a pruning percentage of 75% of the ensemble should allow a good performance of all the methods in the majority of the classification problems. So, all the pruning methods tested in these experiments follow this rule.

5.6.2 Results

Meta-level Results

As mentioned previously, we compared the performance of the meta-model with a baseline using RMSE as error measure. To assess if the meta-model is sig-

nificantly better than the baseline, we used the methodology recommended by Demšar [2006] with $\alpha = 0.05$.

Figure 5.17 shows the Critical Difference (CD) diagrams of the results obtained at the meta-level. For k=20, we can see that both M5' and RF present a better performance that SVM and the baseline. The difference in terms of performance between M5' and RF is not statistically significant. The same can be stated for SVM and the baseline. For k=100, the same result can be verified, although in this case RF shows a slightly better performance than M5'. Again, yet, this difference is not statistically significant.

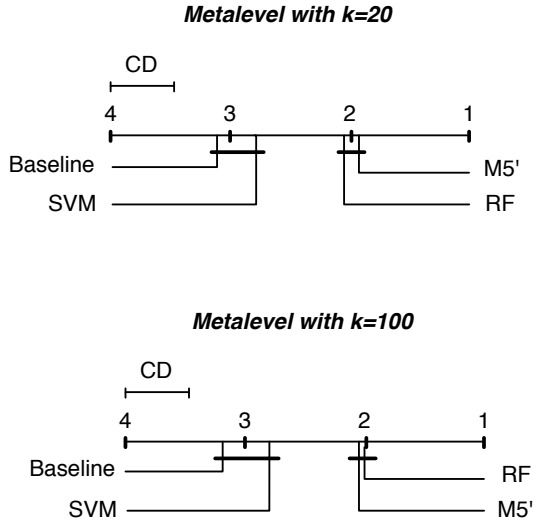

Figure 5.17: Critical Difference diagrams of the performance of the meta-models in comparison with the baseline, at the meta-level.

This result shows that the metafeatures that we generated for this problem are informative and can possibly be used to predict the usefulness of a model generated from a bootstrap sample.

Base-level Results

The base-level evaluation of the method was, again, carried with the methodology recommended by Demšar [2006] with $\alpha = 0.05$.

Figure 5.18 shows the CD diagrams for the base-level results. For k=20, our method achieves the best performance with the M5' learning algorithm (this is

in agreement with the results obtained at the meta-level for k=20). The performance of *Meta.M5'* is worse than *Metatarget*, *Bagging* and *MDSQ*, although this difference is not statistically significant. Comparing with the *Random* benchmark, *Meta.M5'* shows better performance. However, the difference between the generalization ability of the method is not statistically significant.

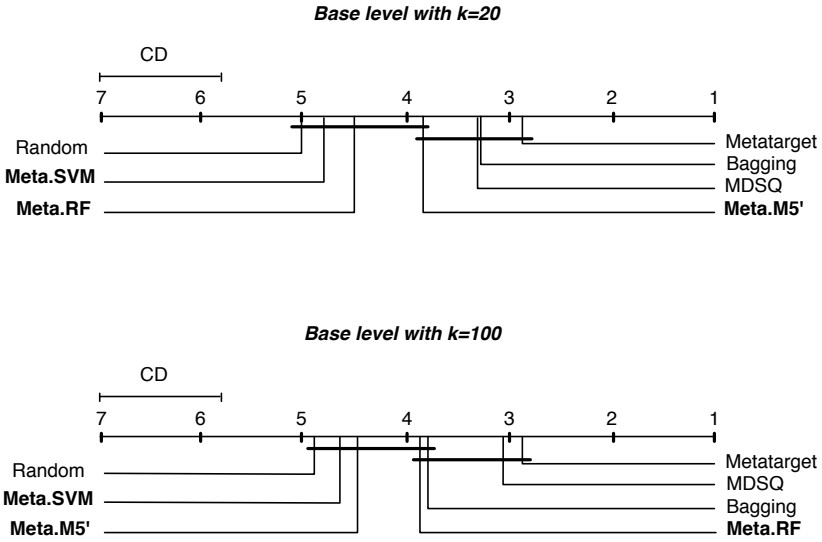

Figure 5.18: Critical Difference diagrams of the performance of the metamodels in comparison with the benchmark pruning methods, at the base-level.

For k=100, our method shows better performance while pairing with the RF learning algorithm (again, this is in agreement with the results obtained at the meta-level for k=20). The results are very similar with the ones obtained in the k=20 scenario. The performance of *Meta.RF* is worse than *Metatarget*, *MDSQ* and *Bagging*, although this difference is not statistically significant. Regarding the comparison with the *Random* baseline, *Meta.RF* presents better performance but the difference is not statistically significant.

The performance of *Bagging* and *MDSQ* is very similar both for k=20 and k=100. *MDSQ* shows a better performance in the k=100 scenario. This result is expected because the method needs a reasonable number of models in order to achieve good performance [Martinez-Muñoz *et al.*, 2009]. This fact also increases the computational cost of the method.

The pruning executed with the *Metatarget* shows the best performance for

both cases, k=20 and k=100. This is indicative that our method, with adequate metafeatures, can become a very useful and innovative pruning technique.

5.6.3 Discussion

The results presented state that our method already has the ability to prune bagged ensembles of decision trees with a performance competitive with the state-of-the-art algorithms. However, there is room for improvement since its performance is not far superior from a naive baseline such as *Random*. Furthermore, since the *Metatarget* benchmark surpasses all the methods tested, it is our goal to further improve the performance of our method in that direction.

We believe that one of the key aspects that affect the most our results is the fact that we are trying to predict the usefulness of one model instead of a subset. In the latter scenario, we could use diversity measures already proposed in the ensemble learning literature as metafeatures and take into account the complementarity between more than two classifiers. The diversity metafeatures that we use in this work such as *COD* or *Q-Statistic* only relate the landmarker models generated from the bootstrap samples with the landmarker models generated from the original training data. We plan to investigate an approach in which our method predicts the accuracy of subsets of models instead of the usefulness of individual ones as we present here.

5.7 Conclusions and Future Work

This chapter proposes a methodology based on an extensive experimental procedure and on MtL for empirically studying the performance of an ensemble learning algorithm, more particularly, bagging. We executed experiments with 53 UCI classification datasets using ensembles of decision trees. Initially, we generated 20 models for each dataset and we tested all possible combinations of those models in the sub-ensemble space. We also executed experiments in which we generated 100 models for each dataset but, due to computational reasons, we were forced to sample the number of combinations tested of the individual models. The results obtained gives us confidence about the effectiveness of the sampling procedure, meaning that it is possible to investigate the distribution of

performance of all bagging ensembles obtained with an algorithm by sampling the results. It would be interesting to repeat the experiments with another base learner such as neural networks but we leave that for future work.

To relate the distribution of performance with the characteristics of the bootstrap samples, we adopted an MtL approach. We used several metafeatures proposed in the literature and we introduce new ones that are very specific of our domain. From our point of view, ensembles are a very promising application of MtL concepts and techniques both to gain a better understanding of their behaviour as well as to develop new ensemble methods.

We focused on understanding the characteristics of bootstraps that generate models that are important for the bagging ensemble. We used exploratory data analysis techniques for that goal. Results show interesting patterns that are discriminative of a bootstrap predictive power 1) the bootstrapping procedure should result in a bootstrap sample that is significantly different from the training set, according to the analysis of the Jensen-Shannon distance; 2) the predictions of a model learned from of a bootstrap should be different from the predictions of a model learned from the training, as is known in the ensemble learning literature. However, this is observed with the COD metric, but not with the Q-Statistic metafeature; 3) the predictive power of a good bootstrap is very similar to the one presented by the training set using naive models.

Regarding the pruning method that we propose, our approach differentiates itself from the other methods proposed in the literature in the sense that enables to prune the ensemble before actually generating the predictive models. This feature can be particularly important in contexts with limited computational resources, such as online applications [Prodromidis and Stolfo, 2001]. We tested the method against bagging and a state-of-the-art pruning technique, *MDSQ*. Results show that our method is competitive with bagging (using only 25 % of the bagged models) and *MDSQ* (with less computational cost since it does not require to generate all the models of the bagged ensemble).

As future work, it would be interesting to investigate an approach in which the MtL method could predict the accuracy of subsets of models instead of the usefulness of individual ones as we introduce here. The motivation for this is that if the metamodel has information about the complementarity of the models,

it could be better informed to make a decision of pruning or not a given model.

Chapter 6

CHADE: Metalearning with Classifier Chains for Dynamic Combination of Classifiers

6.1 Introduction

Ensemble methods are still one of the favourite techniques used by researchers and practitioners to deal with classification problems. The high predictive performance reported together with the increased computational power that we have available has led to a widespread of these techniques. Our proposal focuses on how to aggregate the output of the classifiers in order to achieve a final prediction. We propose a dynamic combination method to combine a subset of classifiers for each test instance. The method is based on a widely known multi-label classification technique, Classifier Chains (CC) [Read *et al.*, 2011].

In a typical classification problem, an instance x, represented by a vector of m attributes values, belongs to only one class, y. However, in a multi-label classification problem, an instance x can belong to a subset of L labels, therefore $Y = \{y_1, ..., y_L\}$. Each label $l, 1 \leq l \leq L$, is associated with x if $y_l = 1$ or not if

$y_l = 0$.

A common technique to deal with multi-label classification problems is the so called *problem transformation*. It consists in decomposing the multi-label problem into several binary problems. Therefore, instead of using a classifier that has the ability to deal with multiple outputs, one binary classifier can be trained for each label. However, this technique has a major drawback: it does not take into account the intrinsic interdependencies that can exist between the labels. This is very important for our problem since it is well known that diversity is a fundamental concept of ensembles [Kuncheva and Whitaker, 2003] and can only be managed by a meta-model that has information about the classifiers interdependencies.

Dynamic classifier selection is the problem of deciding which subset of models from an ensemble should be used to generate the prediction for an instance x. It can be addressed as a multi-label classification problem. An ensemble EH consists of $K, 1 \leq k \leq K$ individual classifiers, $EH = \{h_1, ..., h_K\}$. Each classifier is represented by a label $Y = \{y_{1,x}, ..., y_{k,x}\}, y_{k,x} \in \{0, 1\}$. If $y_{k,x} = 1$, it means that the classifier k correctly classified the instance x; otherwise, $y_{k,x} = 0$. Therefore, using the CC method, we can train a meta-model that relates the attributes of a dataset with the output of each classifier from the ensemble. Given a new test instance x, the meta-model is able to predict which classifiers should be used for the final prediction.

Since our method includes a step in which a model is learned at a higher level (meta), we consider it as a metalearning (MtL) approach [Brazdil *et al.*, 2009]. Therefore, we compare it with other dynamic selection or combination methods (DSC) methods, including an alternative MtL approach [Cruz *et al.*, 2015].

We executed experiments on 42 classification datasets from the UCI repository [Lichman, 2013]. To the best of our knowledge this is largest set of experiments comparing DSC methods. For each dataset, we generated a bagging ensemble of 100 decision stumps. Then, we tested several state-of-the-art DSC methods. Results show the competitiveness of our method, not only at the base-level but also at the meta-level. We also explored the performance of our method in the widely known XOR problem in order to achieve a better understanding

of its behaviour.

The main contributions of this chapter are:

1. new methods for dynamic combination of classifiers, CHADE and E-CHADE
2. an extensive experimental evaluation to demonstrate the competitiveness of CHADE and E-CHADE
3. a novel way of using landmarkers as metafeatures for metalearning

The chapter is organized as follows. In section 6.2 we present a summary of the state-of-the-art for DSC methods. Section 6.3 presents our method for dynamic combination of classifiers. In section 6.4 we describe the experiments that were carried out. Section 6.5 includes a discussion about the characteristics of our method in the light of the results that were obtained in the XOR problem. Finally, section 6.6 concludes the chapter and sets directions of future work.

6.2 Related Work

We organized the state-of-the-art on dynamic approaches for ensembles of classifiers into two groups: dynamic selection, for the methods that only select one classifier for each test instance x; and dynamic combination, for the methods that can select more than one classifier for each test instance.

6.2.1 Dynamic selection

The first paper concerning dynamic selection of classifiers is due to Ho *et al.* [1994]. In that paper, the authors proposed a selection based on a partition of training examples. The individual classifiers are evaluated on each partition to find the best one for each. Then, the test instance to be predicted is categorized into a partition and classified by the corresponding best classifier.

The DS-LA LCA based method and the DS-LA OLA-based method are often used as benchmark in comparative studies [Woods *et al.*, 1997; Britto *et al.*, 2014]. For abbreviation purposes we refer to these as OLA and LCA, respectively. Both methods calculate an estimation of accuracy of the base classifiers in the local region of the feature space close to the test instance in the

training dataset. In OLA, it is the percentage of correct classifications within the local region; in LCA, it is the percentage of correct classifications within the local region but considering only those examples where the classifier has given the same class as the one it gives for the test instance. In both methods, only one classifier is selected for the final prediction.

Concerning MtL approaches, Todorovski and Džeroski [2003] proposed the meta decision trees, a method to select the best predictor of an ensemble of decision trees for a given test instance. The decision is made by a meta-model that learns the prediction patterns of the ensemble summarized in a set of three metafeatures.

Yankov *et al.* [2006] proposed a method to select from an ensemble of two k-NN models the one best suited for a given instance. The selection is done by a Support Vector Machine model using metafeatures extracted from the instances.

6.2.2 Dynamic combination

The first dynamic combination approach was introduced by Merz [1996]. However, the results showed that a simple majority combination was superior to their dynamic approach.

Kuncheva *et al.* [2007] proposed a method based on the oracle concept. Essentially, each classifier of the ensemble consists of two sub-classifiers and an oracle that decides which of the two sub-classifiers is going to be used to predict the test instance. The oracle is a random linear function. They claimed that the random oracle idea works because it adds diversity to the ensemble. Later, Ko *et al.* [2008] improved this idea by adding a k-nearest neighbours approach and proposed the KNORA-E and KNORA-U methods. In the former, only the classifiers that correctly classify all the K-nearest patterns are used; in the latter, the classifiers that correctly classify any of the k-nearest neighbours are used - a single classifier can be selected more than once.

Tsymbal [2000] and Tsymbal and Puuronen [2000] combined dynamic integration with classifier ensembles using bagging and boosting algorithms. Results suggest that dynamic integration improves significantly the performance of the ensembles instead of the more typical majority voting integration. Later, Tsymbal *et al.* [2006] also presented experiments in which a dynamic integration

approach was added to Random Forest, instead of the simple majority voting.

Santana *et al.* [2006] proposed a method that explicitly uses accuracy and diversity to select a subset of classifiers. The method sorts the classifiers in decreasing order of accuracy and in increasing order of diversity. They presented two versions: DS-KNN, which is very similar to LCA and OLA but it takes into account diversity; and DS-Cluster, that uses a clustering process to divide the validation set into clusters and, for each cluster. The most promising classifiers are selected.

Liyanage *et al.* [2013] proposed a dynamically weighted ensemble classification framework whereby an ensemble of multiple classifiers are trained on clustered features. The decisions from these multiple classifiers are dynamically combined based on the distances of the cluster centres to each test data sample being classified. Results showed that their method is significantly better than a Support Vector Machine baseline classifier.

Cruz *et al.* [2015, 2018] proposed a method that uses MtL for dynamic combination of classifiers. This method uses a meta-model to decide if a base classifier is competent to classify a given test instance. The meta-model is learned using metafeatures that capture the prediction patterns of the base classifiers in its regions of competence and in the overall decision space.

Very recently, after the publication of CHADE in [Narassiguin *et al.*, 2017], our method was extended by making use of Probabilistic Classifier Chains [Narassiguin *et al.*, 2017]. This is an indicator that this contribution has indeed open new research directions for the dynamic combination of classifiers.

6.3 CHADE

We propose CHADE (CHAined Dynamic Ensemble) for dynamic combination of ensembles, combining MtL and multi-label learning. The CHADE method, as presented in Figure 6.1, is composed by three stages: 1) generation, 2) meta-training and 3) generalization.

The generation Stage simply consists in training an ensemble of K classifiers. We used bagging [Breiman, 1996a] for experimental purposes, but this is not a requirement of CHADE. For future work, we plan to explore other ensemble

learning algorithms, both for homogeneous and heterogeneous ensembles.

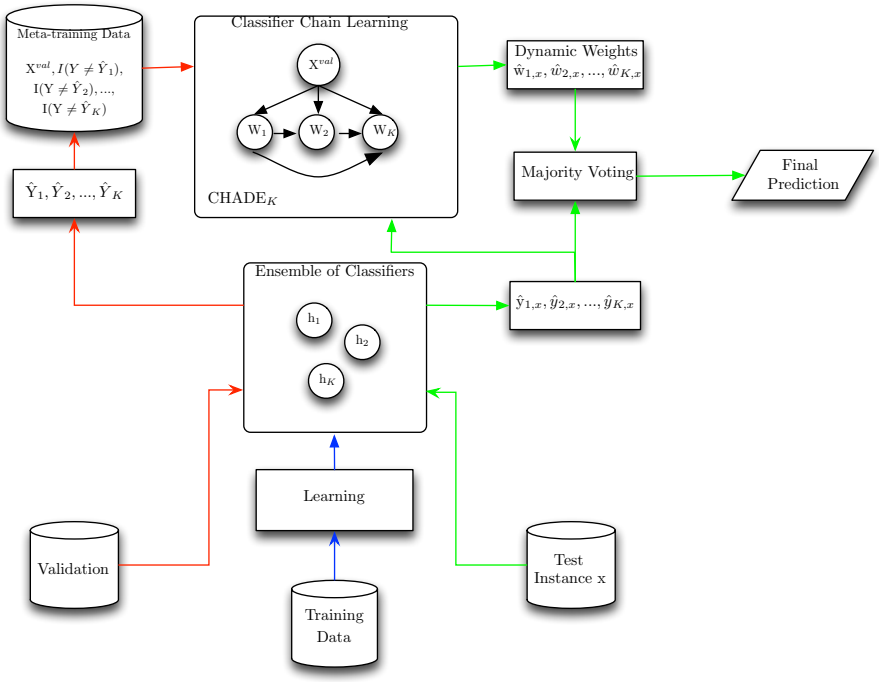

Figure 6.1: CHADE framework.

Stage 2 is the meta-training phase. The ensemble of classifiers $EH = \{h_1, h_2, ..., h_K\}$ is used to make predictions on a validation set. These predictions are then used by a 0-1 loss function, I, that compares them with the true target from the validation set. This generates k binary variables (metafeatures), that represent the performance of each classifier for each example in in the validation set. Therefore, the meta-training data D', is composed by the independent variables of the base-level data, X^{val}; and the K binary variables generated, $W = \{W_1, W_2, ..., W_K\}$.

Note that the extension of the data, W, can be regarded as landmarkers. However, in traditional MtL, landmarkers are aggregated measures of performance of a simple algorithm. In CHADE, since we are interested in capturing patterns at the instance level, it is more useful to not aggregate the performance of the models over all the instances in the validation set. We can also relate this approach with stacking [Wolpert, 1992], since the dataset is extended with information obtained from the predictions made by the base-level classifiers,

Table 6.1: Example of a meta-training dataset *D'*.

X_1	X_2	X_3	W_1	W_2	...	W_K
10	yes	1	0	1	...	1
8	no	2	0	1	...	1
12	no	5	1	1	...	0
...	...	...	...	...	...	...

although in stacking the predictions are used directly and here we use an indicator of the accuracy of the prediction. The meta-training dataset D' shows the morphology of a typical multi-label classification problem, as shown in the example presented in Table 6.1.

After generating the meta-training data, the CC algorithm [Read *et al.*, 2011] for multi-label classification is used for training the meta-model *CHADE*. The pseudo code for Stage 2 is described in Algorithm 3. The training of MC_k, the meta-classifier, can be done with any sort of learning algorithm for classification.

Input: $D = (X^{val}, Y^{val}), EH_k = \{h_1, ..., h_K\}$
Output: $CHADE$
for $k \in 1 \ ... \ |W|$ **do**
  meta-label computation and training
  $D' \leftarrow \{\}$
  for $(x^{val}, y^{val}) \in D$ **do**
    $D' \leftarrow D' \cup ((x, w_1, ..., w_{k-1}), w_k)$
  end
  Train MC_k to predict w_k
  $MC_k : D' \rightarrow w_k \in \{0, 1\}$
end

Algorithm 3: CHADE training pseudocode.

Finally, in Stage 3, the meta-model trained in Stage 2 is used together with the ensemble generated in Stage 1 for a dynamic combination of the classifiers. The pseudocode for this Stage is presented in Algorithm 4. The weight of each classifier for a test instance x is assigned by the meta-model *CHADE*.

The weights $\hat{W}$ are then combined with the base-level predictions $\hat{Y}$ for x by majority voting. This results in the final prediction $\hat{y}$.

Input: $x^{test}, EH_k = \{h_1, ..., h_K\}, CHADE$
Output: $\hat{y}$
$\hat{Y} \leftarrow \{\}$
for $k \in 1 \ ... \ |W|$ **do**
  $\hat{w}_k \leftarrow MC_k$: $(x^{test}, \hat{w}_1, ..., \hat{w}_{k-1})$
  $\hat{Y} \leftarrow \hat{Y} \cup (\hat{w}_k \times \hat{y}_k \leftarrow h_k : (x^{test}))$
end
$MajorityVoting(\hat{Y})$

Algorithm 4: CHADE generalization pseudocode.[1]

CHADE does not require parameter tuning. In comparison with the other DSC techniques that we mentioned in the previous section, CHADE presents a major advantage difference: it does not rely on the nearest neighbours algorithm. This can make CHADE particularly useful in datasets with a large number of training examples, since distance computation can be quite costly in those cases.

The paper that introduced the CC method for multi-label classification also proposed the Ensemble of Classifier Chains (ECC) [Read *et al.*, 2011]. In ECC, several CC models are trained. However, the order in which the classifiers are *chained* is different and each CC model is trained in a bootstrap sample of the training data. In comparison with CC, ECC allows to reduce the variance component of the error. Therefore, we also did experiments with an ensemble version of CHADE that we named E-CHADE.

6.4 Experiments

The experiments that we carried out aimed to answer the following research questions:

1. Can CHADE/E-CHADE improve the performance of Bagging?

[1] When $k = 1$, w_{k-1} takes the form w_0 which we assume to be an empty set for simplicity of the pseudocode provided.

2. How does the performance of CHADE/E-CHADE relates with other Metalearning approaches for dynamic combination of classifiers?

3. How does the performance of CHADE/E-CHADE relates with other state-of-the-art DSC techniques?

6.4.1 Experimental setup

A total of 42 datasets were used in the experiments, all of them obtained from the UCI machine learning repository [Lichman, 2013]. To the best of our knowledge, these are the experiments for dynamic combination of classifiers with the largest number of datasets.

The datasets were split into training (50%), validation (25 %) and test set (25%). The split was done using stratified sampling. Each experiment was repeated 10 times and the results were averaged. We used accuracy as evaluation metric. The methodology proposed by Demšar [2006] was used for statistical validation of the results.

For each learning problem, we generated a bagging ensemble of 100 decision stumps. We chose to use weak learners since it has been reported that this approach enhances the detection of differences between dynamic approaches [Britto *et al.*, 2014].

We selected a few DSC techniques for comparison with our approach. The first is META-DES [Cruz *et al.*, 2015], a MtL method that uses a set of five different meta-features to learn a meta-model that predicts if a base classifier is competent to classify a given test instance. To the best of our knowledge this is the only MtL method for dynamic combination of classifiers proposed in the literature. Our experimental setup is slightly different from the one used by Cruz *et al.* [2015]. Therefore, we made two changes to adapt it to our experimental setup: 1) the meta-model is learned with a decision tree algorithm instead of the Levenberg-Marquadt algorithm. Preliminary results showed that the choice of the learning algorithm for the meta-training did not have a significant impact on the method. Therefore, we decided to use decision trees for the meta-training stage of META-DES and CHADE; 2) one metafeature (perpendicular distance between the input sample and the decision boundary of the base classifier) could not be used since its dependent of the base classifier used in the experiments,

which, originally, was a perceptron classifier and, here, we use decision stumps. The remaining features of the algorithm are implemented as detailed in the original paper. Concerning the parameters of META-DES, we used the ones that achieved the best results in the original experiments.

As for other state-of-the-art DSC techniques, we compare CHADE with OLA, LCA, KNORA-E and KNORA-U. These techniques were selected since they have shown good results in several experimental studies [Britto *et al.*, 2014]. Regarding parameters, k was set to 10 in all experiments. Also, in the base-level experiments, we compare the DSC methods with an abstract model, the Oracle. This model selects the classifier that correctly predicts the label for any given test instance, if such classifier exists. This comparison assesses whether the DSC techniques have room for improvement or not.

Finally, we tested two variations of CHADE: the single meta-model version (*CHADE*) and the ensemble version (*E-CHADE*). The number of meta-models in E-CHADE was set to 10 since it was reported to be a good value for ensembles of classifiers chains [Read *et al.*, 2011]. The details of these methods can be found in section 6.3 of this chapter.

6.4.2 Comparison with another MtL approach

With the aim of a more complete comparison with the alternative approach META-DES, we compared the performance of the MtL methods both at the base-level and meta-level. The purpose of the meta-level evaluation is to assess if the methods are combining the correct models and leaving out the ones that fail the predictions. We also compared the MtL methods with a baseline, that is the majority class in the meta-training data. This is important to evaluate the quality of the combinations that are being made by the MtL approaches.

It is important to notice that a better performance at the meta-level does not necessarily imply a better performance at the base-level. For instance, consider the classifiers h_1, h_2 and h_3, and the meta-models h'_1 and h'_2. Given a test instance x, h'_1 recommends combining h_1 and h_2; on the other hand, h'_2 recommends combining h_1, h_2 and h_3. We then verify that the predictions made by the classifiers are the same for the three of them and they are all correct. This scenario implies that h'_1 has an accuracy of 66.6% and h'_2 has an accuracy of

100%. However, the base-level prediction is the same for both cases. Therefore, h'_1 and h'_2 have the same base-level accuracy.

Figure 6.2 shows the Critical Difference diagram for the comparison of the MtL approaches at the meta-level. E-CHADE clearly presents the best performance. On one hand, the difference between E-CHADE and the other methods evaluated is statistically significant. On the other hand, although CHADE achieves a better mean rank than META-DES and the baseline, there is no evidence in these experiments that the difference is statistically significant.

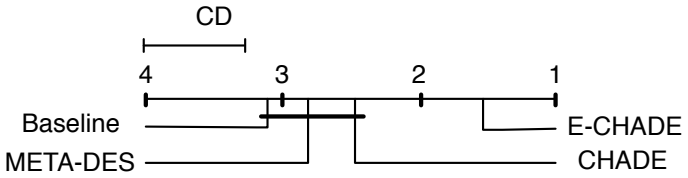

Figure 6.2: Critical Difference diagrams (with $\alpha = 0.05$) for the comparison with META-DES at the meta-level. The null hypothesis of the Friedman's test is rejected for $\alpha = 0.01$, 0.05 and 0.1.

Considering now the base-level performance, Figure 6.3 shows that E-CHADE and CHADE have a better performance than META-DES and the baseline, Bagging. The difference between E-CHADE and CHADE in comparison with Bagging is statistically significant.

These results indicate that the answer to our first question is positive: CHADE and E-CHADE do improve the performance of Bagging.

However, the difference between E-CHADE, CHADE and META-DES is not statistically significant. The results suggest that CHADE and E-CHADE allow an improvement over META-DES but this statement is not statistically validated. The same conclusion can be made for the difference between META-DES and Bagging. This answers the second research hypothesis that we stated previously.

Interestingly, the results obtained at the base-level are in accordance with the ones obtained at the meta-level. In fact, E-CHADE presents the best overall performance, although the difference is more clear for the comparison made at the meta-level. We must also state the Oracle model has indisputably the best

performance of all the methods compared, by far difference. This indicates that there is room for improvement by the dynamic approaches.

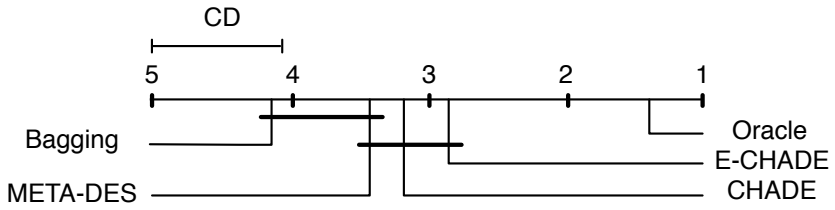

Figure 6.3: Critical Difference diagrams (with $\alpha = 0.05$) for the comparison with META-DES at the base-level. The null hypothesis of the Friedman's test is rejected for $\alpha = 0.01$, 0.05 and 0.1.

6.4.3 Comparison with state-of-the-art

In this section, we extend the comparison made in the previous section to other state-of-the-art DSC methods. The comparison is only made at the base-level since the majority of the methods do not follow a MtL approach.

Figure 6.4 presents the Critical Difference diagram of the experiments. The first result that stands out is that the majority of methods obtain a similar performance. At the top of the ranking, the Oracle model appears isolated; at the bottom, LCA and Bagging appear very close to each other.

A more detailed analysis shows that OLA and E-CHADE present the best performance, followed closely by CHADE and KNORA-E. The difference between these four methods to META-DES and KNORA-U is not statistically significant; however, if we compare them with LCA and Bagging we see that the difference is now statistically significant. Given these results, we find it difficult to extract conclusions. However, the fact that E-CHADE and CHADE are two of the three techniques (excluding the Oracle model) with the best mean ranking is a very promising result.

We must also state that these results are consistent with conclusions in a recent survey about DSC [Britto *et al.*, 2014]. In that survey the authors concluded that there is no evidence that one specific technique may win over all the others for any classification problem. They also suggest that it should be

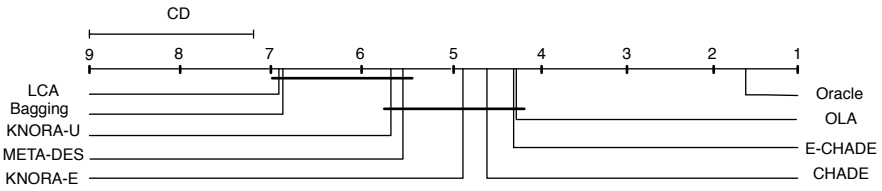

Figure 6.4: Critical Difference diagrams (with $\alpha = 0.05$) for the comparison with several dynamic selection/combination methods at base-level. The null hypothesis of the Friedman's test is rejected for $\alpha = 0.01$, 0.05 and 0.1.

put effort into developing a method that recommends which dynamic approach should be used for a specific dataset. Our results reinforce this claim.

We further investigated the performance of E-CHADE in comparison with CHADE. Specifically, we wanted to verify if the performance of CHADE could be as good as the one obtained by E-CHADE if the meta-model was trained in a specific order. The top plot of Figure 6.5 shows the mean rank as more meta-models are added to E-CHADE; at the bottom, the same figure shows the individual mean rank of each meta-model that is added to E-CHADE. Figure 6.6 presents the same graphs but for the meta-level.

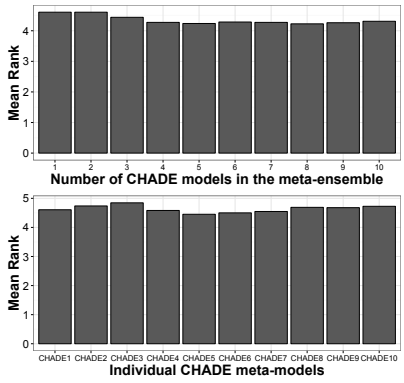

Figure 6.5: Evolution of the base-level mean rank as more meta-models are added to E-CHADE.

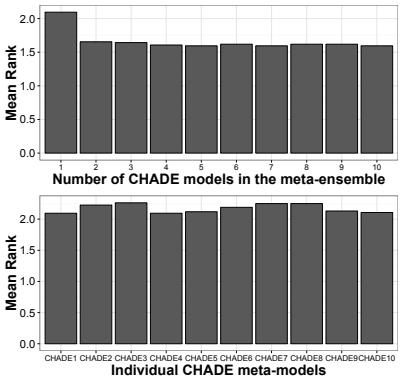

Figure 6.6: Evolution of the meta-level mean rank as more meta-models are added to E-CHADE.

We can see in Figure 6.5 that none of the individual CHADE meta-models achieves a mean rank as good as the one obtained by E-CHADE. This shows that the ensemble framework for classifier chains is effective in improving the

performance of CHADE. This result is consistent with experiments made in multi-label classification datasets at the base-level [Read *et al.*, 2011].

Still regarding Figure 6.5, its possible to verify that the performance of E-CHADE stabilizes after the $4^{\text{th}}/5^{\text{th}}$ meta-model is added to the ensemble. This indicates that E-CHADE does not requires the 10 meta-models that we trained in our experiments and therefore the computational cost of the method can be reduced.

Finally, Figure 6.6 shows the same graphs presented in Figure 6.5 but for the meta-level. The results are quite similar. However, we can see a more pronounced improvement by adding more meta-models to E-CHADE at the meta-level than in the base-level.

6.5 Further Analysis

In this section we discuss and analyse the behaviour of CHADE and E-CHADE in comparison with the Oracle model. For this, we carried out experiments with the XOR problem [Minsky and Papert, 1969]. The XOR problem is a classic example of a dataset in which a linear model will not perform well. As we can see in Figure 6.7, there does not exist a linear model that can separate the blue and red points. Therefore, a decision stump cannot solve this problem. However, a bagging ensemble of decision stumps together with a DSC technique can be used successfully. For that, the dynamic component needs to be able to select the appropriate decision stump(s) for each test instance.

We generated a dataset with 1000 data points of the XOR problem to improve our understanding of CHADE's (and E-CHADE's) behaviour. The aim of this experiment is twofold:

1. Assess if CHADE/E-CHADE is able to correctly identify the appropriate decision stump(s) for each region of the input space.

2. Analyse the combination patterns of CHADE/E-CHADE and compare them with the Oracle.

The XOR data was split into training (50%), validation (25%) and test set (25%) using stratified sampling. Once more, we generated bagging ensembles

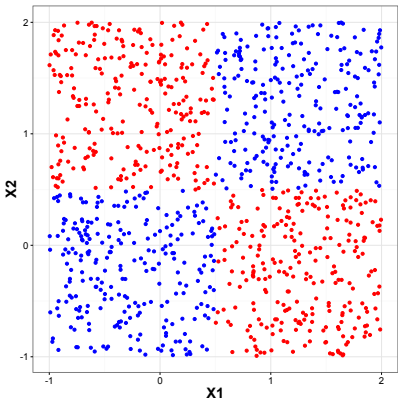

Figure 6.7: XOR problem.

of 100 decision stumps. Both CHADE and E-CHADE present 88% of accuracy in the test set, which greatly improves the 49.6% achieved by Bagging. By definition, as the data is not noisy, the Oracle model achieves 100%.

Figure 6.8 shows three heat maps that represent the combination of classifiers made by CHADE, E-CHADE and Oracle, for each test instance. A red square means that the classifier (column) was selected for the corresponding example (row). We can see that it is possible to identify four regions in the heat maps, each region corresponding to the four regions of the input space visible in Figure 6.7. This result is more clearly seen for CHADE and the Oracle than for E-CHADE. Also, we see that the heat map for CHADE is more similar to the one for the Oracle than the heat map of E-CHADE.

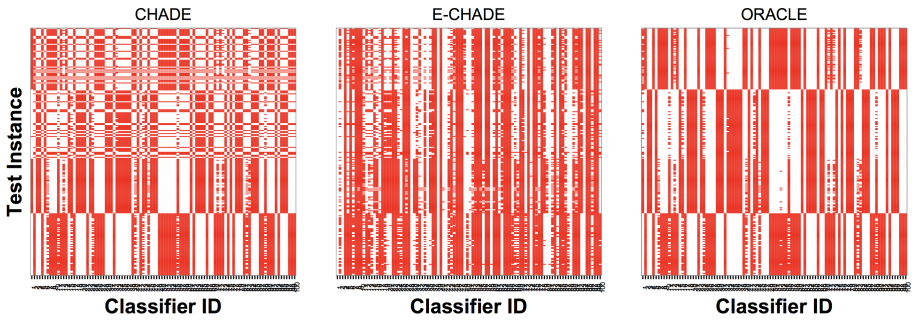

Figure 6.8: Heat maps showing the combination of classifiers made by each technique.

Still regarding Figure 6.8, it seems that E-CHADE's heat map has more red

squares than the other two. This suggests that E-CHADE combines larger sets of classifiers than CHADE or Oracle. We confirmed this result by analysing the distribution of the number of classifiers selected per instance by each method, presented in Figure 6.9; and the distribution of the number of times each classifier was selected, presented in Figure 6.10. E-CHADE selects larger sets of classifiers than the other two; on the other hand, there seems to be no difference between CHADE and Oracle regarding this statistic. Figure 6.10 also shows an interesting result: CHADE and Oracle select the classifiers in a more even manner than E-CHADE. All the classifiers are used at least 120 times by the first two, while E-CHADE often discards some of the classifiers from the generalization phase.

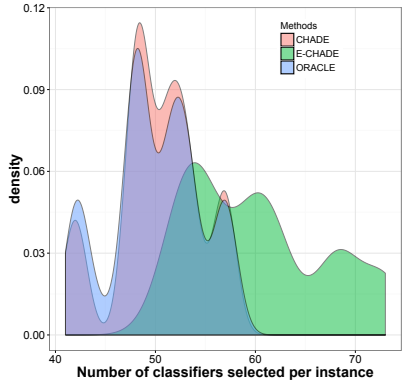

Figure 6.9: Distribution of the number of classifiers selected per instance by each method.

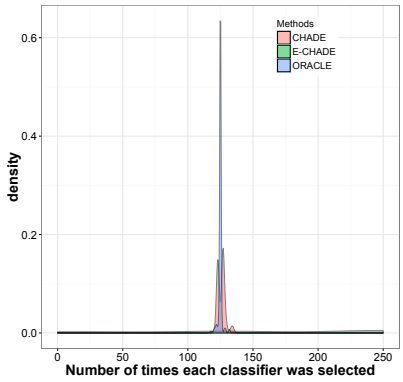

Figure 6.10: Distribution of the number of times each classifier was selected by each method.

Finally, we also computed the percentage of duplicated sets of classifiers combined by each method, shown in Table 6.2. We can see that E-CHADE presents the lower percentage of duplicated classifiers, followed closely by the Oracle model. This result is consistent with the previous one, since we also verified that E-CHADE combines more classifiers per instance than the other methods. Furthermore, this might also indicate that E-CHADE is somehow more *dynamic* than the other techniques, which can be useful in some datasets. We plan to further investigate this characteristic in future work.

Table 6.2: Percentage of duplicated sets of classifiers combined by each method.

CHADE	E-CHADE	Oracle
90%	82.8%	84.4%

6.6 Conclusions and Future Work

This chapter proposes a method for dynamic combination of classifiers that uses the widely known multi-label classification technique, Classifier Chains. In order to do so, we transform the problem of dynamically combining a set of classifiers into a multi-label classification problem. We propose two versions of the method: CHADE and E-CHADE, based on an ensemble variant of CC, ECC.

We evaluated CHADE and E-CHADE in a large set of experiments with 42 classification datasets. Several state-of-the-art DSC techniques were implemented, including one that also uses a MtL approach (META-DES). We tested our methods initially against META-DES and then against several DSC techniques. In the former experiment, the results obtained with CHADE and E-CHADE suggest an improvement over META-DES; in the latter, both CHADE and E-CHADE appear in the top 3 of the techniques with better performance. Given the large number of datasets that were used in the experiments, we consider these results as very promising. We recall that most empirical experiments comparing several DSC techniques show that no method is clearly superior to the others. The characteristics of the learning problem influence a lot the performance of the methods [Britto *et al.*, 2014]. It is expected that as more datasets are used, the smaller the difference between the techniques. Our results reinforce this claim.

For a better understanding of the behaviour of CHADE and E-CHADE, we tested it in the XOR problem. We showed visually that both CHADE and E-CHADE are able to identify the classifiers that are more suitable for each one of the four regions of the input space. We also showed that the combination

patterns obtained by CHADE are more similar to the ones made by the Oracle than the ones created by E-CHADE. This can be justified by the fact that E-CHADE combines, on average, more classifiers than CHADE.

Very recently, CHADE was extended by making use of Probabilistic Classifier Chains [Narassiguin *et al.*, 2017]. This is an indicator that this work has indeed opened new research directions for the dynamic combination of classifiers.

In the future, we plan to study strategies for defining in a non-random way the order in which CHADE and E-CHADE are trained. This could not only improve the performance of the methods but also reduce its computational cost (if less CC's need to be trained). We will start by exploring some strategies already proposed for ECC [Read *et al.*, 2014].

The experiments that we carried out in this chapter were conducted with homogeneous ensembles. We are interested in verifying the performance of CHADE and E-CHADE in a heterogeneous ensembles scenario. Since heterogeneous ensembles are usually more diverse, they should make the task of the dynamic method more difficult. This could also be an opportunity to study the relation between the diversity of an ensemble and the performance of the DSC method.

Finally, DSC methods are one of the techniques that can be used for dealing with concept drift in data streams [Gama *et al.*, 2014]. We plan to test CHADE and E-CHADE for that purpose.

Part IV

Epilogue

Chapter 7

Conclusions

The EL literature states that the field can be split into three topics: generation, pruning and integration. From the perspective of a data scientist that uses EL algorithms, this leads to more decisions that need to be made: how many models to generate? Which pruning technique should be used? Which is the best technique to integrate the individual predictions into a single one? In practice, what happens is that the data scientist ignores most of these questions (mostly due to time constraints) and focus only in tuning the a very small set of hyper-parameters of the EL algorithm that he or she chooses. This can lead to under performance.

In this thesis we aimed to study if MtL can be useful to automate and improve bagging, a widely used EL algorithm. Our research led us to the following contributions:

- A MtL framework for **systematic generation of metafeatures**.
- **New metafeatures** that proved to be informative for the algorithm selection problem in MtL.
- An *autoML* system that combines MtL and a learning to rank approach to **automatically optimize a bagging ensemble**.
- An empirical methodology to study the behaviour of bagging and give insights about the **desired properties of a bagging ensemble**.

- A method that uses **MtL to prune bagging ensembles** by analysing the data characteristics of the bootstrap samples that are generated.
- A **MtL method to dynamically combine a subset of predictors** from an ensemble according to the characteristics of a given test instance.

In the next sections we describe these contributions and how they relate.

Systematic Metafeatures for Metalearning

The selection of metafeatures for MtL is often an *ad hoc* process. The choice of metafeatures is often arguable and can affect the results. In chapter 3, we present a framework to systematically generate metafeatures in the context of MtL. This framework regards a metafeature as a combination of three elements: a meta-function, a tuple of arguments and a post-function. The framework establishes how to systematically generate metafeatures from all possible combinations of arguments and post-functions alternatives that are compatible with a given meta-function. Thus, the development of metafeatures for a MtL approach simply consists of selecting a set of meta-functions (e.g. entropy, mutual information and correlation) and the framework systematically generates the set of metafeatures that represent all the information that can be obtained with those meta-functions from the data.

We carried out experiments on 203 classification datasets in which we compared the sets generated by our approach against a *non-systematic* approach to generate metafeatures and also compared our results against the state-of-the-art. Overall, our results show that the systematically generated metafeatures are more informative than the non-systematic and the state-of-the-art ones.

We also show how the framework can easily generate novel sets of metafeatures that are more informative than the ones generated by common meta-functions used in MtL, such as Pearson's correlation. Specifically, we show that the set of metafeatures generated by MIC, a function to measure non-linear correlation between numeric variables, enable an improvement over the one generated by Pearson's correlation. This is a relevant contribution since Pearson's correlation has been widely used for MtL. Particularly, we show that Maximal Information Coefficient, a measure of non-linear correlation between

numeric variables, can be very useful to characterize datasets.

Automated Machine Learning

Machine Learning (ML) has been successfully applied to a wide range of domains and applications. One of the techniques behind most of these successful applications is EL, a field that gave birth to popular methods such as Random Forests or boosting. The complexity of applying these techniques together with the market scarcity on ML experts, has created the need for systems that enable a fast and easy drop-in replacement for ML libraries. Automated machine learning (*autoML*) is the field of ML that attempts to answer these needs.

Typically, *autoML* systems rely on optimization techniques such as bayesian optimization to search for the best model. In chapter 4, we propose *autoBagging*: an *autoML* system that differentiates itself by making use of a framework for systematic generation of metafeatures (presented in chapter 3) and a learning to rank approach to learn from metadata. *autoBagging* is able to automatically tune a bagging ensemble regarding the number of models to generate, the pruning technique to apply, the percentage of models to prune and the dynamic integration technique to use with the final ensemble.

Empirical results on 140 classification datasets show that *autoBagging* can yield better performance than other state-of-the-art systems such as *auto-sklearn* and achieve results that are not statistically different from an ideal model that always selects the best algorithm for each dataset. For the purpose of reproducibility and generalizability, *autoBagging* is publicly available as an R package on CRAN.

Generation and Pruning of Bagging Ensembles

In chapter 5, we propose and apply a methodology to study the relationship between the performance of bagging and the characteristics of the bootstrap samples. We elaborate on two contributions: 1) an extensive set of experiments to estimate the empirical distribution of performance of the population of all possible ensembles that can be created with a set of bootstraps; 2) a MtL approach to analyse that distribution based on characteristics of the bootstrap samples and their relationship with the complete training set. Our results show

that diversity is indeed crucial for an *relevant* bootstrap and we show evidence of a metric that can measure diversity directly from the bootstrap samples. Specifically, the Jensen-Shannon distance between bootstrap samples.

Furthermore, also in chapter 5, we propose a method that makes use of the metadata collected from the methodology proposed to prune bagging ensembles before generating the individual models. The method is able to prune bagging ensembles based on the characteristics of the bootstrap samples. We carried out experiments in 53 classification datasets for ensembles of 20 and 100 decision trees. Regarding the pruning technique that we propose, our results show that our method can reduce the size of the ensemble to 3/4 of its original size and be competitive both with bagging and a state-of-the-art pruning technique.

Dynamic Integration of Bagging Ensembles

Dynamic selection or combination (DSC) methods select one or more classifiers from an ensemble according to the characteristics of a given test instance x. Most methods proposed for this purpose are based on the nearest neighbours algorithm: it is assumed that if a classifier performed well on a set of instances similar to x, it will also perform well on x.

In chapter 6, we addressed the problem of dynamically combining a pool of classifiers by merging two approaches: MtL and multi-label classification. Taking into account that diversity is a fundamental concept in ensemble learning and the interdependencies between the classifiers cannot be ignored, we solved the multi-label classification problem by using a widely known technique: Classifier Chains (CC). Additionally, we extended the typical MtL approach by combining metafeatures characterizing the interdependencies between the classifiers with the base-level features.

We executed experiments on 42 classification datasets and compared our method with several state-of-the-art DSC techniques, including another MtL approach. Results show that our method allows an improvement over the other MtL approach and is very competitive with the other four DSC methods. Interestingly, this contribution already proven itself worthy of opening new research directions for the field of dynamic combination of classifiers.

7.1 Future Research

The methods described in this thesis can be improved and extended in a number of ways, opening several opportunities for future research. Some ideas are outlined below.

- **Automatic Selection of Meta-functions.** Characterizing a dataset is a hard problem. We believe that the framework that we proposed in chapter 3 is a step towards an easier and more informative characterization of datasets. However, to make it completely automated, the selection of meta-function(s) must also be automated.

- **Metafeature selection.** Although the framework for systematic generation of metafeatures has proven itself very useful, it can be computationally expensive given the large number of metafeatures that often are computed. Can we decrease the set of metafeatures before actually computing them? Two possible approaches to this could be cost-sensitive learning or active learning.

- ***autoBagging* with different base-learners.** The experiments and implementation that we describe in this thesis are based on a version of *autoBagging* that only recommends ensembles of decision trees. The bias of having a single base-learner may be affecting *autoBagging* overall performance. It would be interesting and potentially useful to add other base-learners such as neural networks, which also have been widely used as base-learners for bagging ensembles [Brown, 2004].

- ***autoML* and Explainable AI.** ML models are now widely used in our daily lives. The fact that most of these models are black boxes has raised concerns among some researchers, AI experts and the society as an overall [Gunning, 2017]. This has led to methods being proposed with the purpose of making ML models understandable to humans. One of the research directions in this field is precisely to use ML to explain ML models [Ribeiro *et al.*, 2016]. *autoML* systems may also benefit from this.

- **Automatic Feature Engineering.** *autoML* systems have been proposed for several phases of the KDD process. However, most of them focus on

modelling and/or hyper parameter tuning phase. Feature engineering, one of the most important stages (if not the most) has been often disregarded. Recently, some papers showed efforts towards this [Kanter and Veeramachaneni, 2015; Katz *et al.*, 2016; Lam *et al.*, 2017]. We believe that this is a very promising line of research.

- **Metafeatures for multiple datasets.** Typically, a metafeature characterizes a single dataset (and a meta-example corresponds to a single dataset). How can we design metafeatures that characterize several datasets and how they relate? This could be particularly useful for the problem of pruning bagging ensembles presented in chapter 5: by characterizing several bootstrap samples and how they relate, the metamodel would have information about the complementarity between those bootstrap samples and the models that could be generated.

- **CHADE training.** The *CHADE* algorithm that we propose in chapter 6, designed for dynamic combination of ensembles, uses a Classifier Chain trained with a random order as a metamodel. A promising future line of research would be to study strategies for defining a non-random order in which *CHADE* is trained. This could not only improve the performance of the methods but also reduce its computational cost (if less Classifier Chains need to be trained).

- **CHADE for data streams.** Given the dynamic nature of the data stream scenario, it would be interesting to study how *CHADE* could be used to deal with concept drift and how it could complement a data streams algorithm such as *Leveraging Bagging* [Bifet *et al.*, 2010].

7.2 Publications

The following papers are included in this thesis:

- Pinto, F., Soares, C., & Mendes-Moreira, J. (2014, September). *A framework to decompose and develop metafeatures.* In Proceedings of the 2014 International Conference on Meta-learning and Algorithm Selection-Volume 1201 (pp. 32-36). CEUR-WS. org.

- Pinto, F., Soares, C., & Mendes-Moreira, J. (2014, December). *An empirical methodology to analyze the behavior of bagging.* In Advanced Data Mining and Applications. Springer International Publishing, 2014. 199-212.

- Pinto, F., Soares, C., & Mendes-Moreira, J. (2015, June). *Pruning bagging ensembles with metalearning.* In International Workshop on Multiple Classifier Systems (pp. 64-75). Springer, Cham.

- Pinto, F., Soares, C., & Mendes-Moreira, J. (2016, April). *Towards automatic generation of metafeatures.* In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 215-226). Springer International Publishing.

- Pinto, F., Soares, C., & Mendes-Moreira, J. (2016, September). *CHADE: Metalearning with Classifier Chains for Dynamic Combination of Classifiers.* In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 410-425). Springer International Publishing.

- Pinto, F., Cerqueira, V., Soares, C., & Mendes-Moreira, J. (2017, September). *autoBagging: Learning to Rank Bagging Workflows with Metalearning.* In International Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms co-located with the European Conference on Machine Learning & Principles and Practice of Knowledge Discovery in Databases. CEUR-WS. org.

- Pinto, F., Soares, C., & Mendes-Moreira, J. (2018, January). *Systematic Generation of Metafeatures for Metalearning.* Submitted for publication.

- Pinto, F., Cerqueira, V., Soares, C., & Mendes-Moreira, J. (2018, January). *Automated Bagging Ensembles with Metalearning.* Submitted for publication.

Other publications as first author during the period in which this thesis was developed:

- Pinto, F., Soares, C., & Mendes-Moreira, J. (2014, October). *Simulation of the ensemble generation process: the divergence between data and*

model similarity. In 28th European Simulation and Modelling Conference. Eurosis.

- Pinto, F., Soares, C., & Brazdil, P. (2015). *Combining regression models and metaheuristics to optimize space allocation in the retail industry.* Intelligent Data Analysis, 19(s1), S149-S162.

Bibliography

Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N van Rijn, and Joaquin Vanschoren. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine learning*, 107(1):79–108, 2018.

Shawkat Ali and Kate A Smith-Miles. A meta-learning approach to automatic kernel selection for support vector machines. *Neurocomputing*, 70(1):173–186, 2006.

Shawkat Ali and Kate A Smith. On learning algorithm selection for classification. *Applied Soft Computing*, 6(2):119–138, 2006.

Ethem Alpaydin and Cenk Kaynak. Cascading classifiers. In *Kybernetika*, 1998.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

Eric Bauer and Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine learning*, 36(1-2):105–139, 1999.

Hilan Bensusan and Christophe Giraud-Carrier. Discovering task neighbourhoods through landmark learning performances. In *Principles of Data Mining and Knowledge Discovery*, pages 325–330. Springer, 2000.

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. Leveraging bagging for evolving data streams. *Machine Learning and Knowledge Discovery in Databases*, pages 135–150, 2010.

Catherine Blake and Christopher J Merz. {UCI} repository of machine learning databases. 1998.

Pavel Brazdil, João Gama, and Bob Henery. Characterizing the applicability of classification algorithms using meta-level learning. In *Machine Learning: ECML-94*, pages 83–102. Springer, 1994.

Pavel Brazdil, Carlos Soares, and Joaquim Pinto Da Costa. Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results. *Machine Learning*, 50(3):251–277, 2003.

Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to data mining*. Springer, 2009.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Leo Breiman. Heuristics of instability and stabilization in model selection. *The annals of statistics*, 24(6):2350–2383, 1996.

Leo Breiman. Stacked regressions. *Machine learning*, 24(1):49–64, 1996.

Leo Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):229–242, 2000.

Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

Leo Breiman. Using iterated bagging to debias regressions. *Machine Learning*, 45(3):261–277, 2001.

Alceu S Britto, Robert Sabourin, and Luiz ES Oliveira. Dynamic selection of classifiers—a comprehensive review. *Pattern Recognition*, 47(11):3665–3680, 2014.

Gavin Brown and Ludmila I Kuncheva. "good" and "bad" diversity in majority vote ensembles. In *Multiple Classifier Systems*, pages 124–133. Springer, 2010.

Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

Gavin Brown. *Diversity in neural network ensembles*. PhD thesis, University of Birmingham, 2004.

Gavin Brown. An information theoretic perspective on multiple classifier systems. In *Multiple Classifier Systems*, pages 344–353. Springer, 2009.

Peter Büchlmann and Bin Yu. Analyzing bagging. *Annals of Statistics*, pages 927–961, 2002.

Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. Meta-data: Characterization of input features for meta-learning. In *Modeling Decisions for Artificial Intelligence*, pages 457–468. Springer, 2005.

Vítor Cerqueira, Luís Torgo, Fábio Pinto, and Carlos Soares. Arbitrated ensemble for time series forecasting. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 478–494. Springer, 2017.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM, 2016.

Guilherme P Coelho and FJ Von Zuben. The influence of the pool of candidates on the performance of selection and combination techniques in ensembles. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 5132–5139. IEEE, 2006.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

Louis Columbus. Ibm predicts demand for data scientists will soar 28% by 2020. *Forbes*, 2017.

Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

Rafael MO Cruz, Robert Sabourin, George DC Cavalcanti, and Tsang Ing Ren. Meta-des: A dynamic ensemble selection framework using meta-learning. *Pattern Recognition*, 48(5):1925–1935, 2015.

Rafael MO Cruz, Robert Sabourin, and George DC Cavalcanti. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, 41:195–216, 2018.

Tiago Cunha, Carlos Soares, and André CPLF Carvalho. Metalearning for context-aware filtering: Selection of tensor factorization algorithms. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, pages 14–22. ACM, 2017.

Marcílio Carlos Pereira de Souto, Ricardo Bastos Cavalcante Prudêncio, Rodrigo GF Soares, Daniel SA de Araujo, Ivan G Costa, Teresa Bernarda Ludermir, and Alexander Schliep. Ranking and selecting clustering algorithms using a meta-learning approach. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 3729–3735. IEEE, 2008.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30, 2006.

Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, pages 263–286, 1995.

Pedro Domingos. Why does bagging work? a bayesian account and its implications. In *KDD*, pages 155–158, 1997.

Saso Džeroski and Bernard Ženko. Is combining classifiers with stacking better than selecting the best one? *Machine learning*, 54(3):255–273, 2004.

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, 15(1):3133–3181, 2014.

César Ferri, Peter Flach, and José Hernández-Orallo. Delegating classifiers. In *Proceedings of the twenty-first international conference on Machine learning*, page 37. ACM, 2004.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine

learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

Walter D Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53(284):789–798, 1958.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

Jerome H Friedman and Peter Hall. On bagging and nonlinear estimation. *Journal of statistical planning and inference*, 137(3):669–683, 2007.

Jerome H Friedman. On bias, variance, 0/1—loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1(1):55–77, 1997.

Giorgio Fumera and Fabio Roli. Linear combiners for classifier fusion: Some theoretical and experimental results. In *Multiple Classifier Systems*, pages 74–83. Springer, 2003.

Giorgio Fumera and Fabio Roli. A theoretical and experimental analysis of linear combiners for multiple classifier systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(6):942–956, 2005.

Johannes Fürnkranz and Johann Petrak. An evaluation of landmarking variants. In *Working Notes of the ECML/PKDD 2000 Workshop on Integrating Aspects of Data Mining, Decision Support and Meta-Learning*, pages 57–68, 2001.

Joao Gama and Pavel Brazdil. Characterization of classification algorithms. In *Progress in Artificial Intelligence*, pages 189–200. Springer, 1995.

João Gama and Pavel Brazdil. Cascade generalization. *Machine Learning*, 41(3):315–343, 2000.

João Gama and Petr Kosina. Recurrent concepts in data streams classification. *Knowledge and Information Systems*, pages 1–19, 2013.

João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys (CSUR)*, 46(4):44, 2014.

Mike Gashler, Christophe Giraud-Carrier, and Tony Martinez. Decision tree ensemble: Small heterogeneous is better than large homogeneous. In *Machine Learning and Applications, 2008. ICMLA'08. Seventh International Conference on*, pages 900–905. IEEE, 2008.

Lise Getoor and Lilyana Mihalkova. Learning statistical models from relational data. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 1195–1198. ACM, 2011.

Giorgio Giacinto, Fabio Roli, and Giorgio Fumera. Design of effective multiple classifier systems by clustering of classifiers. In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, volume 2, pages 160–163. IEEE, 2000.

Christophe Giraud-Carrier. The data mining advisor: meta-learning at the service of practitioners. In *Machine Learning and Applications, 2005. Proceedings. Fourth International Conference on*, pages 7–pp. IEEE, 2005.

Taciana AF Gomes, Ricardo BC Prudêncio, Carlos Soares, André LD Rossi, and André Carvalho. Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing*, 75(1):3–13, 2012.

Yves Grandvalet. Bagging equalizes influence. *Machine Learning*, 55(3):251–270, 2004.

David Gunning. Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), nd Web*, 2017.

Mark A Hall. *Correlation-based feature selection for machine learning*. PhD thesis, The University of Waikato, 1999.

Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Morgan Kaufmann, 2006.

Lars Kai Hansen and Peter Salamon. Neural network ensembles. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 12(10):993–1001, 1990.

Daniel Hernández-Lobato, José Miguel Hernández-Lobato, Rubén Ruiz-Torrubiano, and Ángel Valle. Pruning adaptive boosting ensembles by means

of a genetic algorithm. In *Intelligent Data Engineering and Automated Learning–IDEAL 2006*, pages 322–329. Springer, 2006.

Daniel Hernández-Lobato, Gonzalo Martínez-Muñoz, and Alberto Suárez. How large should ensembles of classifiers be? *Pattern Recognition*, 46(5):1323–1336, May 2013.

Tin Kam Ho, Jonathan J. Hull, and Sargur N. Srihari. Decision combination in multiple classifier systems. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(1):66–75, 1994.

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

Frank Hutter, Holger Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. *Learning and Intelligent Optimization*, pages 507–523, 2011.

Md M Islam, Xin Yao, and Kazuyuki Murase. A constructive algorithm for training cooperative neural network ensembles. *Neural Networks, IEEE Transactions on*, 14(4):820–834, 2003.

Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

Alipio M Jorge and Paulo J Azevedo. An experiment with association rules and classification: Post-bagging and conviction. In *Discovery science*, pages 137–149. Springer, 2005.

Alexandros Kalousis and Melanie Hilario. Feature selection for meta-learning. In *Advances in Knowledge Discovery and Data Mining*, pages 222–233. Springer, 2001.

Alexandros Kalousis and Theoharis Theoharis. Noemon: design, implementation and performance results of an intelligent assistant for classifier selection. *Intelligent Data Analysis*, 3(5):319–337, 1999.

Alexandros Kalousis, João Gama, and Melanie Hilario. On data and algorithms: Understanding inductive performance. *Machine Learning*, 54(3):275–312, 2004.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.

Gilad Katz, Eui Chul Richard Shin, and Dawn Song. Explorekit: Automatic feature generation and selection. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 979–984. IEEE, 2016.

Albert HR Ko, Robert Sabourin, and Alceu Souza Britto Jr. From dynamic classifier selection to dynamic ensemble selection. *Pattern Recognition*, 41(5):1718–1731, 2008.

J Zico Kolter and Marcus A Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research*, 8:2755–2790, 2007.

Christian Köpf, Charles Taylor, and Jörg Keller. Meta-analysis: From data characterisation for meta-learning to meta-regression. In *Proceedings of the PKDD-00 Workshop on Data Mining, Decision Support,Meta-Learning and ILP*, 2000.

Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems*, pages 231–238. MIT Press, 1995.

Max Kuhn. Building predictive models in r using the caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86, 1951.

Ludmila I Kuncheva and Christopher J Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine learning*, 51(2):181–207, 2003.

Ludmila Kuncheva, Juan J Rodriguez, et al. Classifier ensembles with a random linear oracle. *Knowledge and Data Engineering, IEEE Transactions on*, 19(4):500–508, 2007.

Ludmila Kuncheva. Switching between selection and fusion in combining classifiers: an experiment. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 32(2):146–156, 2002.

Alexandre Lacoste, Hugo Larochelle, Mario Marchand, and François Laviolette. Sequential model-based ensemble optimization. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, pages 440–448. AUAI Press, 2014.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pages 1–101, 2016.

Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327*, 2017.

Aleksandar Lazarevic and Zoran Obradovic. Effective pruning of neural network classifier ensembles. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 2, pages 796–801. IEEE, 2001.

Jun Won Lee and Christophe Giraud-Carrier. A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841, 2011.

Rui Leite and Pavel Brazdil. Predicting relative performance of classifiers from samples. In *Proceedings of the 22nd international conference on Machine learning*, pages 497–503. ACM, 2005.

Christiane Lemke and Bogdan Gabrys. Meta-learning for time series forecasting and forecast combination. *Neurocomputing*, 73(10):2006–2016, 2010.

Christiane Lemke, Marcin Budka, and Bogdan Gabrys. Metalearning: a survey of trends and technologies. *Artificial intelligence review*, 44(1):117–130, 2015.

Nan Li, Yang Yu, and Zhi-Hua Zhou. Diversity regularized ensemble pruning. In *Machine Learning and Knowledge Discovery in Databases*, pages 330–345. Springer, 2012.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

Hang Li. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94(10):1854–1862, 2011.

M Lichman. Uci machine learning repository, 2013.

Hsuan-Tien Lin and Ling Li. *Infinite ensemble learning with support vector machines*. Springer, 2005.

Jianhua Lin. Divergence measures based on the shannon entropy. *Information Theory, IEEE Transactions on*, 37(1):145–151, 1991.

Yong Liu, Xin Yao, and Tetsuya Higuchi. Evolutionary ensembles with negative correlation learning. *Evolutionary Computation, IEEE Transactions on*, 4(4):380–387, 2000.

Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

Sidath Ravindra Liyanage, Cuntai Guan, Haihong Zhang, Kai Keng Ang, JianXin Xu, and Tong Heng Lee. Dynamically weighted ensemble classification for non-stationary eeg processing. *Journal of neural engineering*, 10(3):036007, 2013.

Dragos D Margineantu and Thomas G Dietterich. Pruning adaptive boosting. In *ICML*, volume 97, pages 211–218. Citeseer, 1997.

G Martinez-Muñoz, Daniel Hernández-Lobato, and Alberto Suárez. An analysis of ensemble pruning techniques based on ordered aggregation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(2):245–259, 2009.

Gonzalo Martínez-Muñoz and Alberto Suárez. Pruning in ordered bagging ensembles. In *Proceedings of the 23rd international conference on Machine learning*, pages 609–616. ACM, 2006.

Kiyotoshi Matsuoka. Noise injection into inputs in back-propagation learning. *Systems, Man and Cybernetics, IEEE Transactions on*, 22(3):436–440, 1992.

William McGill. Multivariate information transmission. *Transactions of the IRE Professional Group on Information Theory*, 4(4):93–111, 1954.

Joao Mendes-Moreira, Jorge Freire de Souse, Alípio M Jorge, and Carlos Soares. An ensemble regression approach for bus trip time prediction. In *Proceedings of the EWGT2006 joint conferences*, 2006.

João Mendes-Moreira, Alipio Mario Jorge, Carlos Soares, and Jorge Freire de Sousa. Ensemble learning: A study on different variants of the dynamic selection approach. In *Machine Learning and Data Mining in Pattern Recognition*, pages 191–205. Springer, 2009.

João Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa. Ensemble approaches for regression: A survey. *ACM Computing Surveys (CSUR)*, 45(1):10, 2012.

Christopher J Merz. Dynamical selection of learning algorithms. In *Learning from Data*, pages 281–290. Springer, 1996.

Claire Clain Miller. Data science: The numbers of our lives. *New York Times*, 2013.

Leandro L Minku and Xin Yao. Ddd: A new ensemble approach for dealing with concept drift. *Knowledge and Data Engineering, IEEE Transactions on*, 24(4):619–633, 2012.

Marvin Minsky and Seymour Papert. *Perceptrons*. MIT press, 1969.

Anil Narassiguin, Haytham Elghazel, and Alex Aussem. Dynamic ensemble selection with probabilistic classifier chains. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 169–186. Springer, 2017.

Sejong Oh. A new dataset evaluation method based on category overlap. *Computers in Biology and Medicine*, 41(2):115–122, 2011.

Julio Ortega, Moshe Koppel, and Shlomo Argamon. Arbitrating among competing classifiers using learned referees. *Knowledge and Information Systems*, 3(4):470–490, 2001.

Yonghong Peng, Peter A Flach, Pavel Brazdil, and Carlos Soares. Decision tree-based data characterization for meta-learning. In *Proceedings of the ECML/P-KDD'02 Workshop on Integration and Collaboration Aspects of Data Mining, Decision Support and Meta-Learning*, pages 111–122, 2002.

Yonghong Peng, Peter A Flach, Carlos Soares, and Pavel Brazdil. Improved dataset characterisation for meta-learning. In *Discovery Science*, pages 141–152. Springer, 2002.

Adam H Peterson and TR Martinez. Estimating the potential for combining learning models. In *Proceedings of the ICML Workshop on Meta-Learning*, pages 68–75, 2005.

Bernhard Pfahringer, Hilan Bensusan, and Christophe Giraud-Carrier. Tell me who can learn you and i can tell you who you are: Landmarking various learning algorithms. In *Proceedings of the 17th international conference on machine learning*, pages 743–750, 2000.

Fábio Pinto, Carlos Soares, and João Mendes-Moreira. Chade: Metalearning with classifier chains for dynamic combination of classifiers. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 410–425. Springer, 2016.

Jordan B Pollack. Backpropagation is sensitive to initial conditions. *Complex Systems*, 4:269–280, 1990.

Andreas L Prodromidis and Salvatore J Stolfo. Cost complexity-based pruning of ensemble classifiers. *Knowledge and Information Systems*, 3(4):449–469, 2001.

Ricardo BC Prudêncio and Teresa B Ludermir. Meta-learning approaches to selecting time series models. *Neurocomputing*, 61:121–137, 2004.

Chao Qian, Yang Yu, and Zhi-Hua Zhou. Pareto ensemble pruning. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 2935–2941, 2015.

J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2012. ISBN 3-900051-07-0.

Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *ICLR*, 2017.

Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

Jesse Read, Luca Martino, and David Luengo. Efficient monte carlo methods for multi-dimensional learning with classifier chains. *Pattern Recognition*, 47(3):1535–1546, 2014.

Matthias Reif, Faisal Shafait, and Andreas Dengel. Meta-learning for evolutionary parameter optimization of classifiers. *Machine learning*, 87(3):357–380, 2012.

Matthias Reif, Faisal Shafait, and Andreas Dengel. Meta2-features: Providing meta-learners more information. In *35th German Conference on Artificial Intelligence*. Citeseer, 2012.

Larry Rendell and Howard Cho. Empirical learning as a function of concept character. *Machine Learning*, 5(3):267–298, 1990.

Larry Rendell, Raj Seshu, and David Tcheng. More robust concept learning using dynamically-variable bias. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 66–78, 1987.

David N Reshef, Yakir A Reshef, Hilary K Finucane, Sharon R Grossman, Gilean McVean, Peter J Turnbaugh, Eric S Lander, Michael Mitzenmacher, and Pardis C Sabeti. Detecting novel associations in large data sets. *science*, 334(6062):1518–1524, 2011.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

John R Rice. The algorithm selection problem. *Advances in computers*, 15:65–118, 1976.

Marko Robnik-Šikonja and Igor Kononenko. Theoretical and empirical analysis of relieff and rrelieff. *Machine learning*, 53(1-2):23–69, 2003.

Juan J Rodriguez, Ludmila I Kuncheva, and Carlos J Alonso. Rotation forest: A new classifier ensemble method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(10):1619–1630, 2006.

Fabio Roli and Giorgio Fumera. Analysis of linear and order statistics combiners for fusion of imbalanced classifiers. In *Multiple Classifier Systems*, pages 252–261. Springer, 2002.

Niall Rooney and David Patterson. A weighted combination of stacking and dynamic integration. *Pattern recognition*, 40(4):1385–1388, 2007.

Niall Rooney, David Patterson, Sarab Anand, and Alexey Tsymbal. Dynamic integration of regression models. In *Multiple Classifier Systems*, pages 164–173. Springer, 2004.

Niall Rooney, David Patterson, Alexey Tsymbal, and Sarab Anand. Random subspacing for regression ensembles. In *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS 2004), Miami Beach, Florida*, 2004.

Bruce E Rosen. Ensemble learning using decorrelated neural networks. *Connection Science*, 8(3-4):373–384, 1996.

André Luis Debiaso Rossi, André Carlos Ponce De Leon Ferreira De Carvalho, Carlos Soares, and Bruno Feres De Souza. Metastream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomputing*, 127:52–64, 2014.

André Luis Debiaso Rossi, Bruno Feres de Souza, Carlos Soares, André de Leon Ferreira de Carvalho, and Carlos Ponce. A guidance of data stream characterization for meta-learning. *Intelligent Data Analysis*, 21(4):1015–1035, 2017.

Carlos Saez, Montserrat Robles, and Juan Miguel Garcia-Gomez. Comparative study of probability distribution distances to define a metric for the stability of multi-source biomedical research data. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*, pages 3226–3229. IEEE, 2013.

Alixandre Santana, Rodrigo GF Soares, Anne MP Canuto, and Marcilio CP de Souto. A dynamic classifier selection method to build ensembles using accuracy and diversity. In *Neural Networks, 2006. SBRN'06. Ninth Brazilian Symposium on*, pages 36–41. IEEE, 2006.

Robert E Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *EEE International Conference on Neural Networks*, pages 407–412. IEEE, 1993.

Klaus Schwab. *The fourth industrial revolution*. Crown Business, 2017.

Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. A survey of intelligent assistants for data analysis. *ACM Computing Surveys (CSUR)*, 45(3):31, 2013.

Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2008.

Carlos Soares and Pavel B Brazdil. Selecting parameters of svm using meta-learning and kernel matrix-based meta-features. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 564–568. ACM, 2006.

Carlos Soares, Pavel B Brazdil, and Petr Kuba. A meta-learning method to select the kernel width in support vector regression. *Machine learning*, 54(3):195–209, 2004.

Carlos Soares. Uci++: Improved support for algorithm selection using datasetoids. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 499–506. Springer, 2009.

Quan Sun and Bernhard Pfahringer. Pairwise meta-rules for better meta-learning-based algorithm ranking. *Machine learning*, 93(1):141–161, 2013.

EK Tang, PN Suganthan, and X Yao. An analysis of diversity measures. *Machine Learning*, 65(1):247–271, 2006.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM, 2013.

Ljupčo Todorovski and Sašo Džeroski. Combining classifiers with meta decision trees. *Machine learning*, 50(3):223–249, 2003.

Alexey Tsymbal and Seppo Puuronen. Bagging and boosting with dynamic integration of classifiers. In *Principles of Data Mining and Knowledge Discovery*, pages 116–125. Springer, 2000.

Alexey Tsymbal, Mykola Pechenizkiy, and Pádraig Cunningham. Dynamic integration with random forests. In *Machine Learning: ECML 2006*, pages 801–808. Springer, 2006.

Alexey Tsymbal. Decision committee learning with dynamic integration of classifiers. In *Current Issues in Databases and Information Systems*, pages 265–278. Springer, 2000.

Kagan Tumer and Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection science*, 8(3-4):385–404, 1996.

Naonori Ueda and Ryohei Nakano. Generalization error of ensemble estimators. In *Neural Networks, 1996., IEEE International Conference on*, volume 1, pages 90–95. IEEE, 1996.

Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. Algorithm selection on data streams. In *Discovery Science*, pages 325–336. Springer, 2014.

Joaquin Vanschoren, Jan N van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Yong Wang and Ian H Witten. Inducing model trees for continuous classes. In *Proceedings of the Ninth European Conference on Machine Learning*, pages 128–137, 1997.

Xiangyang Wang, Jie Yang, Xiaolong Teng, Weijun Xia, and Richard Jensen. Feature selection based on rough sets and particle swarm optimization. *Pattern Recognition Letters*, 28(4):459–471, 2007.

Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series. *Neurocomputing*, 72(10):2581–2594, 2009.

David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

Kevin Woods, W Philip Kegelmeyer Jr, and Kevin Bowyer. Combination of multiple classifiers using local accuracy estimates. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(4):405–410, 1997.

Dragomir Yankov, Dennis DeCoste, and Eamonn Keogh. Ensembles of nearest neighbor forecasts. In *Machine Learning: ECML 2006*, pages 545–556. Springer, 2006.

Yi Zhang, Samuel Burer, and W Nick Street. Ensemble pruning via semi-definite programming. *The Journal of Machine Learning Research*, 7:1315–1338, 2006.

Zhi-Hua Zhou and Nan Li. Multi-information ensemble diversity. In *Multiple Classifier Systems*, pages 134–144. Springer, 2010.

Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1):239–263, 2002.

Zhi-Hua Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC Data Mining and Knowledge Discovery Series. Taylor & Francis, 2012.